VSB — TECHNICAL UNIVERSITY OF OSTRAVA

FACULTY OF ECONOMICS

DEPARTMENT OF FINANCE

Predicting Stock Price Movement with Classification Trees in R
Predikce pohybu ceny akcie pomocí klasifikačních stromů v R

Student: Bc. Zhigang Xu

Supervisor of the diploma thesis: Mgr. Taťána Funioková, Ph.D.

Ostrava, 2020

VŠB - Technical University of Ostrava
Faculty of Economics
Department of Finance

Diploma Thesis Assignment

Student:	**Bc. Zhigang Xu**
Study Programme:	N6202 Economic Policy and Administration
Study Branch:	6202T010 Finance
Title:	Predicting Stock Price Movement with Classification Trees in R Predikce pohybu ceny akcie pomocí klasifikačních stromů v R
The thesis language:	English

Description:

1. Introduction
2. Stock Price Movement Prediction Using Classification Trees
3. Prediction Model Implementation in R
4. Stock Price Movement Prediction for Selected Company
5. Conclusion

Bibliography
List of Abbreviations
Declaration of Utilisation of Results from the Diploma Thesis
List of Annexes
Annexes

References:

BREIMAN, L., J. H. FRIEDMAN, R. A. OLSHEN and C. J. STONE. *Classification and regression trees.* New York: Chapman & Hall, 1993. ISBN 978-0-412-04841-8.
KIRKPATRICK, D. Charles and Julie, R. DAHLQUIST. *Technical analysis: the complete resource for financial market technicians.* 3rd ed. Old Tappan: Pearson Education, 2016. ISBN 978-0-13-413704-9.
LANTZ, Brett. *Machine learning with R: discover how to build machine learning algorithms, prepare data, and dig deep into data prediction techniques with R.* 2nd ed. Birmingham: Packt Publishing, 2015. ISBN 978-1-78439-390-8.

Extent and terms of a thesis are specified in directions for its elaboration that are opened to the public on the web sites of the faculty.

Supervisor: **Mgr. Taťána Funioková, Ph.D.**

Date of issue: 22.11.2019
Date of submission: 24.04.2020

Ing. Iveta Ratmanová, Ph.D.
Head of Department

doc. Ing. Lenka Kauerová, CSc.
Vice Dean for Study Affairs
delegated to the negotiation r.no.
VSB/19/050319/9900 from 24.9.2019

The declaration

“Herewith I declare that I elaborated the entire thesis, including all annexes, independently”

Ostrava dated 29.05.2020

徐志刚 Xu Zhigang

……

Student’s name and surname

Content

1 Introduction

It is well-known that stock market price series are generally dynamic in making investments risky. In addition, we are mindful of the fact that stock market price movement is considered to be a random process with fluctuations that are more prominent in the short run. Traders are more likely to buy a stock in the current period whose value is expected to increase in the future and conversely for falling prices. It is straightforward that accurate prediction of the trends in stock market prices maximizes capital gains and minimizes losses.

This work presents the use Machine Learning (ML) techniques to predict stock prices at the level of a firm to get better insights into the accuracy of price movements. The goal is to design an intelligent model that learns from the market data using machine learning techniques and predicts the direction of a changeable stock price.

The thesis is divided into five chapters and the structure is as follows. The first chapter is an introduction that focuses on explaining the main goals of this thesis and the structure of the thesis.

Chapter 2 simply describes stock market theory and different types of technical indicators. We also briefly describe some machine learning algorithms and we emphasize the CART decision tree methodology which is the only one we use to classify and predict the data. We also introduce the notation of the CART process and explain the algorithm and idea of cross-validation.

Chapter 3 is devoted to the prediction model implementation in R. We just briefly describe programming language R and R studio. This chapter includes a list of helpful packages and related functions we will use in the application part to build an optimal model to describe the model and evaluate its performance.

Chapter 4 shows an application for classification and prediction. In this chapter, we simply introduce the company together with the data we used to analyze. In the growing phase, we describe the way of splitting for the first split in detail and describe how to build the maximal tree. In pruning phase, we use cross validation to get optimal tree and tree with minimal cross-validation error. As for the accuracy of the prediction, we show how the classes can be predicted and how the confusion matrices can be

generated. This part focuses on a five-day window. At the end, it presents results for another three time windows, namely a one-, 30-, and 90-day time windows.

Finally, the last chapter is conclusion. It includes summary of our prediction results for all four time windows.

2. Stock Price Movement Prediction Using Classification Trees

2.1 Stock Market

It is a challenge and complex task for investors to predict the stock price because the price can be influenced by many variables like economic recession, product demand, ruling political, investor's sentiment and political events. The correct prediction of a stock future price can make a great return.

There exist two common ways of predicting stock market. First one is future price of a stock and another is based on predicting the future price movement of a stock. The aim of this work is not the accurate value of the stock price but the direction of movement of stock price. The problem is that stock price's time series have noisy feature which refers to the inability to obtain complete information from past behaviors of financial markets to get the dependence between future prices and past prices.

In response to the difficulty of traditional forecast models like time series model, machine learning techniques have been applied for this prediction. Unlike early models with a certain degree of accuracy, machine learning focus on the historical data sets that are analyzed with data mining methods. We mainly adopted one of supervised learning algorithms called classification and regression tree (CART) algorithm.

For a long time, it was believed that changes in the prices of stocks are not forecastable. The reason is the well-known Efficient Market Hypothesis (Jensen, 1978), which states that a market is efficient with respect to a current information if it is impossible to make capital gains in this market. This hypothesis indicates that the current market price discloses all available information. Especially in the strongest form of EMH, historical prices follow a random walk pattern and the future prices cannot be predicted. However, many facts show that EMH is not completely correct but some false. And many traders or investors rely on technical or fundamental analysis of past information to predict the stock prices.

They also buy when the market is in "bullish" and sell when it is "bearish", thus there is exactly correlation between the current and future trends.

2.1.1 Fundamental and Technical analysis

Fundamental and technical analysis are two completely different methods under making investment decision on stocks.

Fundamental analysis evaluate the value of a stock by using public data which is accessible in a company's financial statements such as revenues, net income, return on equity, asset and liability, cash flow and other data to influence a company's current value.

Technical analysis is the idea that the investments based on the price and volume movements of stocks by using candlestick chart charts and other tools including various technical indicators, investors trade only on price momentum but ignore the fundamental characteristics of a company.

2.1.2 Technical Indicators

Technical indicators are important tools that are calculated from time series stock data that aim to forecast financial market movement. These instruments are widely used by investors to check for bearish or bullish signals. The close price is a most common used indicators, and it can also be high, low, open, or average price.

Technical indicators are divided into three main different types, including trend, price, and volume indicators. You can find various indicators in the list of books used (Kirkpatrick,2016). Only those indicators that will be used in the present model are introduced in this chapter.

Trend indicators

Trend indicators tell us which direction the market is moving in, if there is a trend at all. We will discuss Moving Average.

Simple Moving Average

The most commonly used type of moving average is the simple moving average (SMA). Moving Averages are used to smooth the data in an array to help eliminate noise and identify trends. SMA is literally the simplest form of a moving average. Each output value is the average of the previous n values. In a SMA, each value in the time period

carries equal weight, and values outside of the time period are not included in the average. This makes it less responsive to recent changes in the data, which can be useful for filtering out those changes. We choose n equal to 5,10,20 days, and those 3 numbers will be use in practical part. SMA can be expressed as:

$$SMA(n)_t = \frac{\sum_{i=t-n+1}^{n} price_i}{n} \tag{2.1}$$

Weighted Moving Average

The Weighted Moving Average (WMA) calculates a weight for each value in the series. The WMA reacts more quickly to price changes than the regular SMA. The more recent values are assigned greater weights. We can use the following formula of WMA:

$$WMA(n)_t = \frac{price_t \cdot n + price_{\mathrm{t}-1} \cdot (n-1) + price_{t-2} \cdot (n-2) \ldots price_{\mathrm{t}-(n-1)} \cdot 1}{n} \tag{2.2}$$

Exponential Moving Average

The Exponential Moving Average (EMA) is a staple of technical analysis and is used in countless technical indicators. In a SMA indicator, each value in the time period carries equal weight, and values outside of the time period are not included in the average. However, the EMA indicator is a cumulative calculation, including all data. Past values have a diminishing contribution to the average, while more recent values have a greater contribution. This method allows the moving average to be more responsive to changes in the data. The equation for EMA is as follows:

$$EMA(n)_t = EMA(n)_{t-1} + k \cdot (input_t - EMA(n)_{t-1}) \tag{2.3}$$

where input represent price or another technical indicator and k is a constant factor between 0 and 1. For an n-day EMA value of $k = \frac{2}{(n+1)}$ is commonly used.

Moving Average Convergence Divergence

The Moving Average Convergence Divergence (MACD) signals trend changes and indicates the start of new trend direction. High values indicate overbought conditions, low values indicate oversold conditions. Divergence with the price indicates an end to the current trend, especially if the MACD is at extreme high or low values. When the

MACD line crosses above the signal line a buy signal is generated. When the MACD crosses below the signal line a sell signal is generated. To confirm the signal, the MACD should be above zero for a buy, and below zero for a sell.

The MACD indicator is the difference between two EMAs, short EMA based on 12 days period and long EMA based on 26 days period. The Signal line is an Exponential Moving Average of the MACD. The formula for MACD is as follows:

$$MACD_t = EMA(12)_t - EMA(26)_t \tag{2.4}$$

where

$$EMA(12)_t = \frac{2}{13} \cdot price_t + \frac{11}{13} \cdot EMA(12)_{t-1} \tag{2.5}$$

where

$$EMA(26)_t = \frac{2}{27} \cdot price_t + \frac{25}{27} \cdot EMA(26)_{t-1} \tag{2.6}$$

Price indicators

Price indicators show how strong the trend is and can also tell you if a reversal is going to occur. They can be useful for picking out price tops and bottoms. It include Momentum, Rate of Change, Relative Strength Index and Stochastic.

Momentum

The Momentum is a measurement of the acceleration and deceleration of prices during on n-days period. It indicates if prices are increasing at an increasing rate or decreasing at a decreasing rate. The Momentum function can be applied to the price, or to any other data series. The Momentum can be computed as:

$$Momentum(n)_t = price_t - price_{t-n} \tag{2.7}$$

Rate of Change

The Rate of Change (ROC) function measures rate of change relative to previous periods. The function is used to determine how rapidly the data is changing. The

function can be used to measure the ROC of any data series, such as price or another indicator. When used with the price, it is referred to as the Price Rate of Change and for n-days period we have:

$$ROC(n)_t = \frac{price_t}{price_{t-n}} \tag{2.8}$$

Relative Strength Index

The Relative Strength Index (RSI) measures the strength of an issue against its history of price change by comparing "up" days to "down" days. This index is based on the assumption that overbought levels generally occur after the market has advanced for a disproportionate number of days, and that oversold levels generally follow a significant number of declining days. The RSI measures a security's strength relative to its own price history, not to that of the market in general. Because of its name, a common misconception is that this indicator compares one security with other securities. To construct the RSI, several calculations must be made as follows:

$$\text{RS} = \frac{UPS}{DOWNS} \tag{2.9}$$

where UPS is average sum of gains over n periods, DOWNS is average sum of loss over n periods. The standard is to use 14 days period. For instance, RSI could be constructed as:

$$\text{RSI} = 100 - [\frac{100}{1 + RS}] \tag{2.10}$$

Stochastic Oscillator

The Stochastic Oscillator measures where the close is in relation to the recent trading range. The values range from zero to 100. The %K Oscillator is also replaced as Fast %K, and which is generally considered too erratic to use for crossover signals. The Stochastic Oscillator is specified as follows:

$$\%K_t = 100 \cdot \frac{close_t - lowest\ low_{[last\ n\ periods]}}{Highest\ High_{[last\ n\ periods]} - lowest\ low_{[last\ n\ periods]}} \tag{2.11}$$

where the standard is to use 14 days period.

For instance, slow %D values over 75 indicate an overbought condition; values under 25 indicate an oversold condition. It can be expressed as:

$$\%D_t = SMA(3)_t\ of\ \%K_t \tag{2.12}$$

Williams %R

The Williams %R (WPR) is similar to an unsmoothed Stochastic %K. The values range from zero to 100, and are charted on an inverted scale, that is, with zero at the top and 100 at the bottom. Values below 20 indicate an overbought condition and a sell signal is generated when it crosses the 20 line. Values over 80 indicate an oversold condition and a buy signal is generated when it crosses the 80 line. This indicator can be stated in a formula as:

$$\%R_t = 100 \cdot \frac{Highest\ High_{[last\ n\ periods]} - close_t}{Highest\ High_{[last\ n\ periods]} - lowest\ low_{[last\ n\ periods]}} \quad (2.13)$$

where the standard is to use 14 days period.

Williams Accumulation/Distribution

Williams Accumulation/Distribution indicator measures market pressure. Look for divergence with price. When the price makes a new low, but the AD does not, look for the price to turn up, and vice versa.

To calculate Williams' Accumulation/Distribution indicator, first determine the True Range High ("TRH") and True Range Low ("TRL"):

$$TRH_t = \max(close_{t-1}, high_t), \quad (2.14)$$

$$TRL_t = \min(close_{t-1}, low_t), \quad (2.15)$$

where $close_{t-1}$ is yesterday's close and $high_t$ and low_t are today's high and low respectively.

The Williams' Accumulation/Distribution indicator depends on the relationship between today's closing price ($close_t$) to yesterday's closing price.

If today's close is greater than yesterday's close:

$$A/D_t = close_t - \text{TRL} + A/D_{t-1} \quad (2.16)$$

If today's close is less than yesterday's close:

$$A/D_t = close_t - \text{TRH} + A/D_{t-1} \quad (2.17)$$

If today's close is equal to yesterday's close:

$$A/D_t = A/D_{t-1} \quad (2.18)$$

Commodity Channel Index

Commodity Channel Index (CCI) is designed to detect beginning and ending market trends. The range of 100 to -100 is the normal trading range. CCI values outside of this range indicate overbought or oversold conditions. If the price is making new highs, and the CCI is not, then a price correction is likely. The CCI can be obtained using the following equation:

$$CCI_t = \frac{TP_t - ATP_t}{0.015 \cdot MD_t} \quad (2.19)$$

where TP means Typical Price, it is calculated by $\frac{high_t + low_t + close_t}{3}$, ATP means $SMA(14)_t \ of \ TP,$ MD means Mean Deviation of TP, which can be computed as $\frac{\sum_{i=t-n+1}^{n} |TP_t - ATP_t|}{\mathrm{n}}$.

Volume indicators

Volume indicators tell us how volume is changing over time, how much of stock are being bought and sold over time. This is useful because when the price changes, the volume gives an indication of how strong the move is. Bullish moves on high volume are more likely to be maintained than those on low volume.

On Balance Volume

The On Balance Volume (OBV) is a cumulative total of the up and down volume. When the close is higher than the previous close, the volume is added to the running total, and when the close is lower than the previous close, the volume is subtracted from the running total.

To interpret the OBV, look for the OBV to move with the price or precede price moves. If the price moves before the OBV, then it is a non-confirmed move. A series of rising peaks, or falling troughs, in the OBV indicates a strong trend. If the OBV is flat, then the market is not trending.

For instance, the OBV could be constructed as:

If $close_t > close_{t-1}$ then

$$OBV_t = OBV_{t-1} + \text{volume} \quad (2.20)$$

If $close_t < close_{t-1}$ then

$$OBV_t = OBV_{t-1} - \text{volume} \tag{2.21}$$

If $close_t = close_{t-1}$, then

$$OBV_t = OBV_{t-1} \tag{2.22}$$

2.2 Classification Trees in Machine Learning

As we all know, stock market price sequence is usually dynamic, nonparametric, noisy, which makes investment have inherent risk. In addition, in view of the model specifications to be followed in the near future, we should bear in mind the fact that stock price changes are considered to be random processes, and their volatility is more important in the short term. Needless to say, an in-depth understanding of recent stock price trends should help minimize this risk. Therefore, it is very direct to predict the trend of stock market price accurately, which can maximize capital gains and minimize losses.

It is better to recognize that it is not easy to add value to this complex and deeply researched topic, especially considering that millions of data points are being generated around the world in each time period under consideration. Therefore, we introduce the use of machine learning (ML) technology not to predict company's stock prices, but in order to better understand the accuracy of price movements.

ML algorithms can be divided into two main categories: supervised learners for building prediction models and unsupervised learners for building description models. Common supervised learning algorithms include for example Naive Bayes Classifier, Neural Networks, Support Vector Machines and Decision Trees. Which type we need to use depends on the learning task we want to complete.

A predictive model is used for tasks that involve the prediction of one value using other values in the dataset. The learning algorithm attempts to discover and model the relationship among the target feature (the feature being predicted) and the other features. Because predictive models are given clear instruction on what they need to learn and how they are intended to learn it, the process of training a predictive model is known as supervised learning. The often-used supervised machine learning task of predicting which nominal ordinal category an example belongs to is known as classification. It is easy to think of potential uses for a classifier (Lantz, 2015).

In this thesis, we use Decision Trees for classification. There is a lot of alternative algorithms for Decision Tree building, such as ID3, C4.5, CHAID, or Conditional

Inference Trees.

The ID3 (Iterative dichotomiser 3) algorithm was first proposed by Quinlan (1996). The algorithm is based on information theory and takes information entropy and information gain as the measurement standards, so as to realize the inductive classification of data. In case of features with continuous values, ID3 is not effective.

The C4.5 algorithm is an extension of the ID3 algorithm. This algorithm uses information gain ratio to select attributes, it can also process non-discrete data and have ability to process incomplete data. But it cannot run when the training dataset is too large to fit in memory. The need to perform multiple sequential scans and sorts on the data set results in inefficient algorithms.

The CHAID (Chi-squared Automatic Interaction Detection) is a decision tree technique using Person's Chi-square tests of independence. It uses multiway splits and it's suitable for classification as well as regression trees. Conditional Inference Trees are using permutation-based significance tests instead. But both CHAID and Conditional Inference Trees split only if there is a significant association and so prevent overfitting.

Classification and Regression Tree (CART) is a common supervised machine learning technique that can be applied to predict either a categorical target variable, producing a classification tree, or a continuous target variable, producing a regression tree. Most commonly, CART is applied to binary classification or regression.

There are several factors that can affect algorithm selection, such as the data structure, distribution of features, presence of unbalance data, but also software availability or its limitations. We can list some advantages of CART algorithm supporting our choice:

- The series of if-else conditions are highly interpretable and so the model is very intuitive,
- Nonlinear relationship among features has no effect on its performance,
- It can handle both numerical and categorical data or missing values, and does not require normalization or scaling of input data,
- It is nonparametric method and it involves both testing with a testing data and

cross-validation to assess the goodness of fit.

Any machine learning task can be broken down into a series of more manageable steps. It can be organized according to the following process:

1. Collecting data: It is the most important step in solving any supervised machine learning problem. This data will serve as the learning material an algorithm uses to generate actionable knowledge.

2. Exploring and preparing the data: The quality of any machine learning project is based largely on the quality of data it uses. This step in the machine learning process tends to require a great deal of human intervention. Understanding the characteristics of your data beforehand will enable you to build a better model. This could simply mean obtaining a higher accuracy. Before our data can be fed to a model, it needs to be transformed to a format the model can understand.

3. Building a model on the data: The rpart programs build classification models of a very general structure, the resulting models can be represented as binary trees. There are some restrictions in rpart package concerning depth of the maximal tree.

4. Evaluating model performance: Because each machine learning model results in a biased solution to the learning problem, it is important to evaluate how well the algorithm learned from its experience. Depending on the type of model used, we might be able to evaluate the accuracy of the model using a testing dataset, or we may need to be developed measures of performance specific to the intended application.

5. Improving model performance: If better performance is needed, it becomes necessary to utilize more advanced strategies to augment the performance of the model. Sometimes, it may be necessary to switch to a different type of model altogether. We may need to supplement your data with additional data or perform additional preparatory work as in step two of this process

We will describe the steps in the context of stock price movement prediction with CART algorithm.

2.2.1 Collecting and Preparing Data

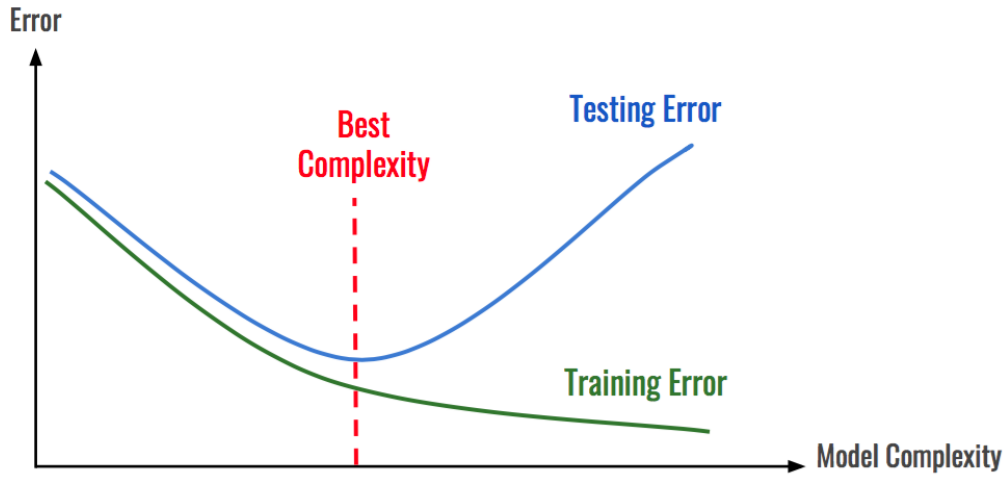
Historical stock quotes can be downloaded from several websites for free to collect data, e.g. from Yahoo! Finance: https://finance.yahoo.com/.

As soon as we gathered the data we have to choose the target variables (target features) and independent variables (target features) with respect to predict stock movement. Typically, some data transformation or feature extraction is needed in this step. It is exactly what we do calculation of a technical indicator for feature extraction.

ML algorithms can produce overly complex models with results of particular changes in the data, or without control. A model that fits the training data too well will typically not predict well using new data. So this problem is known as overfitting which means that the fitted algorithm does not generalize well to new data.

In Figure 2.1, if the model still overfit the training dataset which means that the model is too powerful. We have to reduce the complexity and error of the model.

Figure 2.1 Overfitting

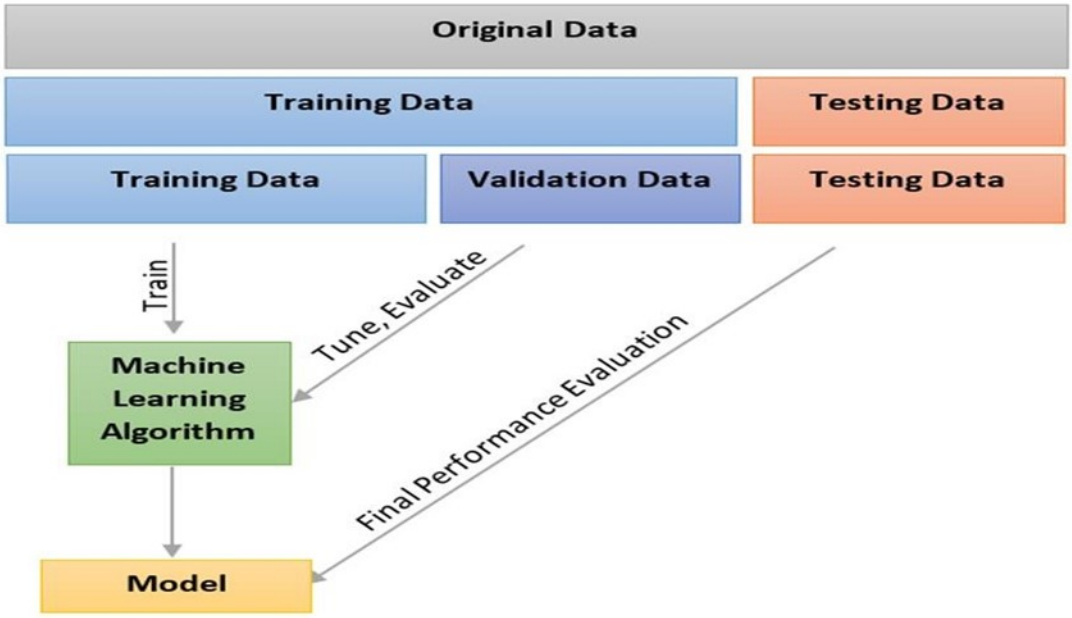


Source:https://hackernoon.com/memorizing-is-not-learning-6-tricks-to-prevent-overfitting-in-machine-learning-820b091dc42

It is important to note the division of the data set in order to describe generalization and overfitting of an ML model. The data set is typically divided into three non-overlapping samples: 1) training sample used to train the model; 2) validation sample for evaluation the model; and 3) test sample for testing the model's performance evaluation to predict well on new data. There are some techniques which use training data for validation as well. e.g. k-fold cross-validation.

In Figure 2.2, the original data is divided like this structure. In practice model, it is typical to use 70% of the samples for training and 30% for testing.

Figure 2.2 Data Set



Source:https://medium.freecodecamp.org/how-to-get-a-grip-on-crossvalidations-bb0ba779e21c

The nature of our data together with its level of measurement and distribution led to the choice of CART algorithm. CART is also composed of feature selection, tree generation and pruning, and can be used for classification.

If the model fits data well, it means that accuracy is high. (Scott,2018) For decision trees, maximal tree fits the training data too well. We cannot use it for new or testing data with the same high accuracy, because maximal tree does not fit testing data well. This is the reason why we want to find optimal tree. It does not fit the testing data so well as maximal tree does, but this optimal tree is applicable to new data.

2.2.2 Building a CART Model

First phase is growing to generating tree which output is maximal tree.

Second phase is pruning, the output is a set of trees, which from a root node to the maximal tree. Their number depends on number of values of complexity parameter alpha.

Last phase is cross-validation, the output is the tree with minimal CV error but what is more important is the optimal tree based upon 1-SE rule.

2.2.2.1 Growing Phase

I focus on CART algorithm here. That is, we have some predictors (features) X_1, X_2, ..., X_n, these can be both categorical and numerical. Suppose the cases fall into J classes: C_1, ..., C_J. The construction of a classifier is based on a learning sample of N cases. Every case x from the learning data set can be represented by a vector of predictor variables and its true class.

Learning data set (N cases):

$$(x_{11}, x_{12}, ..., x_{1n}, j_1)$$
$$\vdots$$
$$(x_{N1}, x_{N2}, ..., x_{Nn}, j_N)$$

Let N_j be number of cases in class j, then proportion N_j/N, j =1, ..., J is a relative frequency of class C_j. A population proportion of class C_j, referred to as the prior class probability, is denoted by π_j.

For given a classifier τ — classification rule represented by a classification tree— which is a collection of decision nodes connected by branches, extending from the root node and terminating at the leaf nodes.

Let A be a node, then τ(A) is a class assigned to node A by the classifier τ, N_A is number of observations in node A, and N_j(A) the number of class j cases in A.

Probability of node A for future observations is then estimated by the formula

$$\mathrm{P(A)} = \sum_{j=1}^{J} \pi_j \frac{N_j(A)}{N_j}, \qquad (2.23)$$

Similarly, given node A, the probability of class *i* in the node A is estimated by

$$P(i|A) = \pi_i \frac{N_i(A)}{N_i} / \sum_{j=1}^{J} \pi_j \frac{N_j(A)}{N_j}. \quad (2.24)$$

When we set the prior probabilities π_j equal to observed class relative frequencies in the sample then

$$\mathrm{P(A)} = \sum_{j=1}^{J} \frac{N_j}{N} \cdot \frac{N_j(A)}{N_j} = \sum_{j=1}^{J} \frac{N_j(A)}{N} = \frac{N_A}{N}, \quad (2.25)$$

and

$$P(i|A) = \frac{N_i}{N} \cdot \frac{N_i(A)}{N_i} / \frac{N_A}{N} = \frac{N_i(A)}{N_A}. \quad (2.26)$$

Given a classifier, that is, given a function τ taking values from {C1, … , CJ}, we want to see how accurate the classifier is. We estimate the probability that τ will misclassify a new case with the risk of tree. Firstly, we need to define the risk of node A:

$$\mathrm{R(A)} = \sum_{i=1}^{n} p(i|A) L\big(i, \tau(\mathrm{A})\big), \quad (2.27)$$

where $\tau(\mathrm{A})$ is a class assigned to node A to minimize this risk and where L is a loss matrix, that means $L(i, \tau(\mathrm{A}))$ is the loss of assigning class $\tau(\mathrm{A})$ to an case which actually belongs to class i. If we split node A into two children A_L and A_R, then

$$\mathrm{P}(A_L) \cdot R(A_L) + \mathrm{P}(A_R) \cdot R(A_R) \leq P(A) \cdot R(A). \quad (2.28)$$

(this is proven in Breiman 1984). The risk of a classifier τ, represented by tree T is then given by

$$\mathrm{R}(\tau) = R(T) = \sum_{i=1}^{n} P(T_i) R(T_i), \quad (2.29)$$

where T_1, T_2, ..., T_n are the terminal nodes of a tree T.

The main difference between CT algorithms is in the so-called split criterion, that is, how to find the best split of node A into two children A_L and A_R. In CART algorithm we assess the goodness of split criterion in terms of impurity. It based on criterion which makes each branch is as pure as possible. It is measured by an impurity function f_{imp} ($\mathrm{p_1}$ . . . $\mathrm{p_J}$). We use Gini impurity (another function, information gain based upon entropy is used in e.g. ID3 or C4.5 algorithm).

$$f_{Gini}(p_1, \dots p_j) = \sum_{i=1}^{J} p_i \cdot (1 - p_i) = 1 - \sum_{i=1}^{J} {p_i}^2. \quad (2.30)$$

That is, impurity of a node *A* is given by

$$I(A) = \sum_{i=1}^{C} P(i|A) \cdot \big(1 - P(i|A)\big) = 1 - \sum_{i=1}^{C} P(i|A)^2. \quad (2.31)$$

If we split node *A* into two children A_L and A_R, then we focus on a decrease in impurity

$$\Delta(A, A_L, A_R) = I(A) - \mathrm{P}(A_L) \cdot I(A_L) - \mathrm{P}(A_R) \cdot I(A_R). \quad (2.32)$$

We select the split that maximize $\Delta(A, A_L, A_R)$. That is, for every predictor X_1, X_2, ..., X_n, we generate possible splits and calculate the difference $\Delta(A, A_L, A_R)$. A split maximizing $\Delta(A, A_L, A_R)$ is called the primary split (the best split). The second, third, etc. split (using other predictors) are called surrogate splits. Those surrogate splits can be used to classify cases with missing data and can be used to assess the variable importance of every predictor.

Variable importance

Given that one of the goals of CART is to develop a simple tree structure for predicting data, relatively few variables may appear explicitly as splitters, which might be interpreted to mean that the other variables are not important in understanding or predicting the dependent variable. However, unlike a linear regression model, a variable in CART can be considered highly important even if it never appears as a node splitter. Because CART keeps track of surrogate splits in the tree-growing process, the contribution a variable can make in prediction is not determined only by primary splits.

To calculate a variable importance score, CART looks at the improvement measure attributable to each variable in its role as either a primary or a surrogate splitter. The values of all these improvements are summed over each node, then scaled relative to the best performing variable. A variable can obtain an importance score of zero in CART only if it never appears as either a primary or a surrogate splitter. Because such a variable plays no role anywhere in the tree, eliminating it from the data set should make no difference to the results.

The importance score measures a variable's ability to perform in a specific tree of a specific size either as a primary splitter or as a surrogate splitter. It says nothing, however, about the value of the variable in the construction of other trees. As a tree is allowed to become bigger, variables have more opportunities to play a role in the tree and thus to receive non-zero importance scores. The relative importance rankings of variables can change dramatically as you compare trees of substantially different sizes. Thus, we should not take importance scores to indicate an absolute information value of a variable; the rankings are strictly relative to a given tree structure.

2.2.2.2 Pruning Phase

Pruning reduces the size of decision trees by removing sections of the tree that provide little hard to classify instances. It also reduces the complexity of the final classifier, and hence improves predictive accuracy by the reduction of overfitting.

The pruning of the decision tree is often achieved by minimizing the loss function or cost function of the decision tree as a whole. This step is also called cost-complexity pruning.

We have built a complete tree, possibly quite large and/or complex, and must now decide how much of that model to retain. In stepwise regression, for instance, this issue is addressed sequentially and the fit is stopped when the F test fails to achieve some level α.

Let T_1, T_2, ..., T_n be the terminal nodes of a tree T. Define |T| to be a number of terminal nodes and the risk of T was introduced in (2.27).

In comparison to regression, |T| is analogous to the degrees of freedom and R(T) to the residual sum of squares.

Now let α be some number between 0 and ∞ which measures the cost of adding another variable to the model; α will be called a complexity parameter. Define

$$R_\alpha(T) = (T) + \alpha \cdot |T| \tag{2.33}$$

to be the cost for the tree and define T_α to be that sub tree of the full model which has minimal cost. Obviously, T_0 is the full model because we want to minimize only R(T):

$$R_0(T) = (T) + 0 \cdot |T| = R(T), \tag{2.34}$$

and so $R(T_0)$ denotes the risk for the full model. Similarly, T_∞ is the model with no splits at all.

The complexity parameter is used to control the size of the decision tree and to select the optimal tree size. If the cost of adding another variable to the decision tree from the current node is above the value of given complexity parameter, then tree building does not continue. We could also say that tree construction does not continue unless it would decrease the overall lack of fit by a factor of given complexity parameter.

Generally speaking, the larger alpha is, the stronger the pruning are, the smaller the optimal subtree is compared with the original decision tree.

The pruning phase selects models (subtrees) with the smallest complexity cost with respect to possible values of complexity parameter (α). The larger the subtree, the better the fit with the training data, but the more complex the model is. For different values of α only limited set of models is generated. As a consequence, we can split interval [0 ,∞) into intervals of the form $(\alpha_{i-1}, \alpha_i]$:

$$[0, \infty) = [0, \alpha_1] \cup (\alpha_1, \alpha_2] \cup \ldots (\alpha_{m-2}, \alpha_{m-1}] \cup (\alpha_{m-1}, \infty), \qquad (2.35)$$

where for any $\alpha \in (\alpha_{i-1}, \alpha_i]$ the same subtree $T_\alpha = T_{\alpha_i}$ is obtained and thus also for value of $\beta_i = \sqrt{\alpha_{i-1} \cdot \alpha_i}$, i=2,,,m-1, we have $T_{\beta_i} = T_{\alpha_i}$. Finally, we set $\beta_1 = 0$ and $\beta_m = \infty$ to complete the sequence of $\beta_1, \ldots, \beta_m$. As a result of pruning phase we have *m* models. All those models are candidates for the optimal model.

In the next step we select the complexity parameter value for pruning the tree which has the lowest cross validation error.

Cross Validation

Cross Validation is a practical way to statistically cut data samples into smaller subsets. The basic idea of cross-validation is to group the original datasets in a sense, one part as a train set and the other part as a validation set or test set.

An improved method is to further divide the new training set and validation set on the basis of the training set. In the new training set training model, in the validation set test model, the initial model is constantly adjusted, so that the best performance of the training model in the validation set is achieved, and finally the evaluation result of the optimal model is obtained on the test set.

Generally speaking, the more training data can reflect the real distribution of data, the better the effect of model training, the more likely to get unbiased estimation. So the idea of cross validation came into being and it can make full use of all training data for evaluation model.

10-fold cross-validation

In general, K-fold cross-validation means that the initial sample is divided into K sub-samples, a single sub-sample is retained as the data of the verification model, and the other K-1 samples are used for training. The cross-validation is repeated K times, and each sub-sample is verified once. The result of averaging K times or using other combination methods finally results in a single estimate. The advantage of this method is that the randomly generated sub-samples are repeatedly used for training and verification, and the results are verified once each time. We need to recollect that a lower value of *K* increases bias and a higher value of *K* increases variance and 10-fold cross-validation is the most commonly used.

The Training data set is randomly divided into 10 subsets of the same size (as nearly as possible). Then the data is trained in 9 of 10 subsets. Then the remaining subset is for testing the model. For each of these iterations, we evaluate that individual error which is also means accuracy. After that we just simply take the average (see Figure 2.3). The advantage to this method is that we can see the error in all aspects of data instead of just testing on one specific subset of it. Then we can know which error is the smallest one. An estimator of true misclassification rate of the tree:

$$\mathrm{E} = \frac{1}{10}\sum_{i=1}^{10} E_i \ . \tag{2.36}$$

Figure 2.3 10-fold cross validation

Source: http://www.bubuko.com/infodetail-2459508.html

1-SE rule Cross validation

One approach is to choose the model that gives the absolute minimum of the cross-validated error. If the fitting has been repeated several times, the cross-validation errors can be averaged over the separate runs such as 10-fold cross-validation.

A more cautious approach is to choose a model is 1-SE rule cross-validation where there is some modest certainty that the final split is giving a better than chance improvement. The 1-SE rule for tree post-pruning is based on the cross-validation estimates of the error of the sub-trees of the initially grown tree, together with the standard errors of these estimates. These values are used to select the final tree model.

As we mentioned, an estimator of true misclassification rate based on cross-validation is given by (2.56). We have to calculate this estimate for every value of $\beta_1, \dots, \beta_m$ obtained from pruning phase. The estimator of true misclassification rate of T_β is given by

$$R_{CV}(\beta) = \frac{1}{10}\sum_{i=1}^{10} R\left(T_\beta^{(i)}\right), \tag{2.37}$$

where $R\left(T_\beta^{(1)}\right), \dots, R\left(T_\beta^{(10)}\right)$ are misclassification errors resulting form 10-fold cross-validation applied on T_β. The standard error of this estimator is given by

$$SE_{CV}(\beta) = \sqrt{\frac{R_{CV}(\beta)(1 - R_{CV}(\beta))}{N}}. \tag{2.38}$$

Firstly, we are looking for β_{min} from $\{\beta_1, \dots, \beta_m\}$ with the smallest cross-validation error $R_{CV}(\beta)$:

$$R_{CV}(\beta_{min}) = \min_{i=1,\dots,m} R_{CV}(\beta_i). \tag{2.39}$$

Sometimes subtree $T_{\beta_{min}}$ is used as the optimal model. But keep in mind that the choice of β_{min} can be affected by the seed of the random number generator to separate learning data set into 10 subtests. Thus, the 1-SE rule is often applied to select the right sized tree: For our purpose, the optimal tree, $T_{\beta_{opt}}$, is the tree corresponding to β_{opt}

where β_{opt} is the maximum β from $\{\beta_1, \ldots, \beta_m\}$ satisfying:

$$R_{CV}(\beta_{opt}) \leq R_{CV}(\beta_{min}) + SE_{CV}(\beta_{min}). \tag{2.40}$$

Finally, we choose β_{opt} to be the optimal complexity parameter and $T_{\beta_{opt}}$, to be the optimal model.

2.2.3 Evaluating Model Performance

In general, any data can be used as a training data to build a model. But only valid models should be used for future prediction or classification. So before we start with its application we have to discuss and evaluate its performance. If we obtain promising results based upon cross-validation or testing data set, then we can expect a good prediction or classification ability also for new data. There are many ways how the performance can be evaluated (Abhijit, 2017). We use mainly confusion matrices and derived characteristics.

2.2.3.1 Confusion Matrix

Confusion matrices are calculated using the predictions of a model on a data set. From confusion matrix, you can gain a better understanding of the strengths and weaknesses of your model, and you can better compare two alternative models to understand which one is better for your application. Traditionally, a confusion matrix is calculated using a model's predictions on a testing dataset.

In Table 2.1, each column of the matrix represents an instance in the predictive class, and each row represents an instance in the actual class. And each class divided into positive and negative. A confusion matrix is a table that categorizes predictions according to whether they match the actual value in the data. The outputs can be represented in a 2x2 matrix shown below:

Table 2.1 Confusion matrix

		Predicted class	
		P	***N***
Actual Class	***P***	True Positives (TP)	False Negatives (FN)
	N	False Positives (FP)	True Negatives (TN)

Source: https://jkmsmkj.blogspot.com/2018/10/confusion-matrix.html

Note that the entries inside of a confusion matrix (TPs, FPs, FNs, TNs) are counts:

- True Positives (TPs): the number of positive cases that the model correctly classified as positive.
- True Negatives (TNs): the number of negative cases that the model correctly classified as negative.
- False Positives (FPs): the number of negative cases that the model incorrectly classified as positive.
- False Negatives (FNs): the number of positive cases that the model incorrectly classified as negative.

The accuracy of a classification is measure in terms of ACC, defined as:

$$ACC = \frac{TP+TN}{\mathrm{TP+FP+FN+TN}}, \tag{2.41}$$

which helps to know what percentage of the predictions are correct.

The sensitivity of a model (also called recall, hit rate or true positive rate) measures the proportion of positive cases that were correctly classified which as shown in the following formula:

$$\mathrm{TPR} = \frac{TP}{(TP + FN)}. \tag{2.42}$$

The specificity of a model (also called the true negative rate), measures the proportion of negative cases that were correctly classified. As with sensitivity, this is computed as the number of true negatives divided by the total number of negatives—the true negatives plus the false positives:

$$\mathrm{TNR} = \frac{TN}{(\mathrm{TN} + \mathrm{FP})}. \tag{2.43}$$

The precision (also known as the positive predictive value) is defined as the proportion of positive cases that are truly positive. A precise model will only predict the positive class in cases very likely to be positive:

$$\mathrm{PPV} = \frac{\mathrm{TP}}{(\mathrm{TP} + \mathrm{FP})}. \tag{2.44}$$

The negative predictive value is defined as the proportion of positive cases that are truly negative. A precise model will only predict the negative class in cases very likely to be negative:

$$\text{NPV} = \frac{\text{TN}}{(\text{TN} + \text{FN})}. \tag{2.45}$$

F-score is a measure of model performance that combines precision and recall into a single number. The F-measure combines precision and recall using the harmonic mean. The harmonic mean is used rather than the more common arithmetic mean since both precision and recall are expressed as proportions between zero and one. The following is the formula for F_1:

$$F_1 = \frac{2 \cdot \text{PPV} \cdot \text{TPR}}{\text{PPV} + \text{TPR}} = \frac{2 \cdot TP}{2 \cdot TP + FP + FN}. \tag{2.46}$$

The kappa statistic adjusts accuracy by accounting for the possibility of a correct prediction by chance alone. Kappa values range to a maximum value of 1, which indicates perfect agreement between the model's predictions and the true values, but which is a rare occurrence. Values less than one indicate imperfect agreement. The definition of kappa is:

$$\text{kappa} = \frac{\Pr(a) - \Pr(e)}{1 - \Pr(e)}, \tag{2.47}$$

where Pr refers to the proportion of actual (a) and expected (e) agreement between the classifier and the true values.

Depending on how your model is to be used, the interpretation of the kappa statistic might vary. One common interpretation is shown as follows（Landis,1996）:

- Poor agreement = Less than 0.20
- Fair agreement = 0.20 to 0.40
- Moderate agreement = 0.40 to 0.60
- Good agreement = 0.60 to 0.80
- Very good agreement = 0.80 to 1.0

3. Prediction Model Implementation in R

In the thesis, we use R studio (RStudio Team, 2016) and R program language (R Core Team, 2020) to solve given classification problem. Let us briefly introduce the R project and the packages we would use in the application.

3.1 R program and R Studio

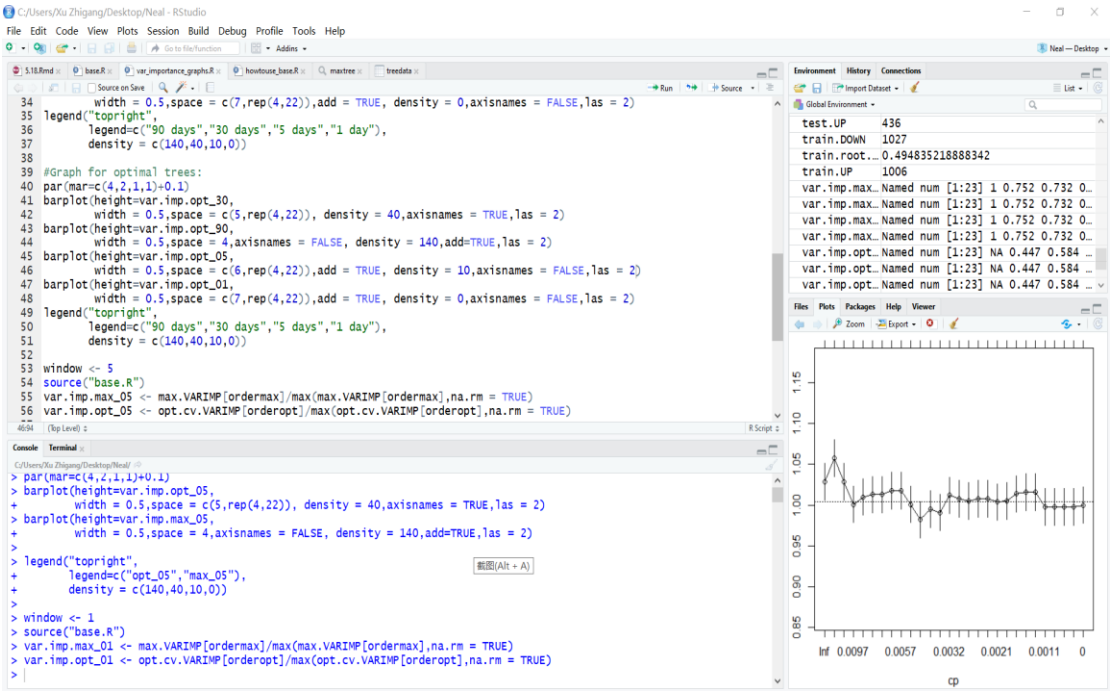
R is a programming language and totally free software environment for statistical computing and graphics supported by the R Foundation for Statistical Computing. The R language is widely used among statisticians and data miners for developing statistical software and other analysis for data sets. R is highly extensible through the use of user submitted packages for specific functions or special areas of study. Due to its S heritage, R has stronger object-oriented programming facilities than the other statistical computing languages. Extending R is also eased by its lexical scoping rules. Another strength of R is static graphics, which can produce publication-quality graphs, including mathematical symbols. Dynamic and interactive graphics are available through additional packages.

R Studio is a free and open-source integrated development environment for R, a programming language for statistical computing and graphics. That is, to build a classification tree and calculate prediction results.

We can download R and R Studio from internet.
(https://cran.r-project.org/ and https://www.rstudio.com/products/rstudio/download/)

A screenshot from the R studio is shown in Figure 3.1.

Figure 3.1 The screenshot of the R studio



The top left quadrant is the editor. It is where we write R code you want to keep for later – functions, packages, etc. This is identical to every other code editor's main window. You can also create a Word document (or an HTML or PDF file) with R Markdown here.

The lower left quadrant is the console. It's a Read–eval–print loop (REPL) for R in which we can test out the ideas, datasets, filters, and functions. This is where we verify an idea we had works before copying it over into the editor above.

The top right quadrant has three important tabs: environment, history and connections. Environment refers to the console environment (see above) and will list, in detail, every single symbol you defined in the console (whether via sourcing or directly). History lists every single console command you executed since the last project started. It is saved into a hidden ".Rhistory" file in your project's folder.

The bottom right panel contains five separate tabs. The first one, Files, is self-explanatory. The Plots tab will contain the graphs you generated with R. It is there you can zoom, export, configure and inspect your charts and plots. The Packages tab lets you install additional packages into R and a brief description is next to each available

package. The Help tab lets you search the incredibly extensive help directory and will automatically open whenever we call help on a command in the console. Finally, the Viewer is essentially RStudio's built-in browser.

If you want to use any additional packages, like rpart: Recursive Partitioning and Regression Trees (Therneau and Atkinson, 2019) you have to install it and load it. As we mentioned, packages can be found in Packages tab. We can also use function `install.packages( )` or `library( )` to install or load a package like that:

```
install.packages("rpart")

library("rpart")
```

3.2 Selected Packages and Functions in R

We introduce some essential packages with their functions which we use in this thesis. Some packages depend on other packages, we do not list those dependent packages if it is not necessary. In this work R version 4.0.0 (2020-04-24) was used.

3.2.1 Data Collection and Preparation

The most common way of storing data in R and the most often data structure used for data analyses is a data frame. It is a two dimensional data structure where columns represent features and rows represent cases. We also use an extension of `data.frame` called `data.table` (Dowle and Srinivasan, 2019) offering a faster development. Obviously, because we explore times series in this work, we also use package `xts` (eXtensible Time Series), and extension of package zoo aimed at irregular time series of numeric vector or matrices and factors.

There are two essential packages for time series analysis `TTR`, and `Quantmod`, which depends on TTR.

Package `quantmod` (Ryan and Ulrich, 2020): its full name is Quantitative Financial Modelling and Trading Framework for R, which is designed to assist the quantitative trader in the development, testing, and deployment of statistically based trading models. We use function `getSymbols()` from this package to load our data.

It is a function to load and manage Symbols in specified environment. We use only three parameters in this work

```
getSymbols(Symbols = ..., from = ..., to = ... )
```

Symbols represent a character vector specifying the names of each symbol to be loaded, representing our company and from and to specify the time period in question. The data is downloaded from the internet, so if we want to work offline we can use functions `saveRDS`() and `readRDS`() to save and read downloaded '`xts`' object locally as an `.rds` file (such a file for our data is attached to this thesis). After download the data, if we need a tool to create standard financial charts given a time series like object. We use function `chartSeries()` to create a historical closing price chart.

In target variable, we use function `lag(x, k = 1, ...)` from package xts, which computing lags and differences on xts objects. And because we need this feature to be a factor (see function `rpart()` below) to assign class with help of function `as.factor`(). By default, when applied to a data frame, it only affects labelled columns.

For feature extraction package TTR (Ulrich, 2019) was used. The Technical Trading Rules package contains many of the most popular technical analysis functions (see Table 3.1), as well as functions to retrieve U.S. stock symbols, and data from Yahoo Finance.

Table 3.1 Technical indicators and R functions

Names	Formulas	Function
SMA	$\frac{\sum_{i=t-n+1}^{n} price_i}{n}$	SMA( )
WMA	$\frac{price_t \cdot n + price_{t-1} \cdot (n-1) \ldots price_{t-(n-1)} \cdot 1}{n}$	WMA( )
EMA	$EMA(n)_{t-1} + k \cdot (input_t - EMA(n)_{t-1})$	EMA( )
MACD	$EMA(12)_t - EMA(26)_t$	MACD( )
Momentum	$Momentum(n)_t = price_t - price_{t-n}$	momentum( )
ROC	$ROC(n)_t = \frac{price_t}{price_{t-n}}$	ROC( )
RSI	$100 - [\frac{100}{1 + \frac{UPS}{DOWNS}}]$	RSI( )
Williams A/D	$A/D_t = close_t$ - max/$min(close_{t-1}, high_t)$+ A/D_{t-1}	williamsAD()
CCI	$\frac{TP_t - ATP_t}{0.015 \cdot MD_t}$	CCI( )
OBV	$OBV_t = OBV_{t-1}$+volume	OBV( )
Stochastic %K	$100 \cdot \frac{close_t - lowest\ low_{[last\ n\ periods]}}{Highest\ High_{[last\ n\ periods]} - lowest\ low_{[last\ n\ periods]}}$	stoch( )
Stochastic %D	$\%D_t = SMA(3)_t\ of\ \%K_t$	stoch( )
Williams %R	$100 \cdot \frac{Highest\ High_{[last\ n\ periods]} - close_t}{Highest\ High_{[last\ n\ periods]} - lowest\ low_{[last\ n\ periods]}}$	WPR( )

The following is the first three functions of MAs:

- `SMA(x, n = 10, ...)`
- `WMA(x, n = 10, wts = 1:n, ...)`
- `EMA(x, n = 10, wilder = FALSE, ratio = NULL, ...)`

where x is the closing price, n means the number of periods to average over. `wts` means vector of weights in WMA. And wilder means if TRUE, a Welles Wilder type EMA

will be calculated and ration is a smoothing or decay ratio. We have to control only x and n; default values of the other parameters will generate indicators as defined above.

The function MACD() compares a fast moving average of a series with a slow mowing average of the same series and it can be used as a generic oscillator for any univariate series, not only price.

```
MACD(x,nFast=12,nSlow=26,nSig=9,maType,percent=TRUE, ...)
```

We use the function in this form to calculate MACD introduced …….

```
MACD(x, nFast = 12, nSlow = 26, nSig=4, maType=EMA, percent
=FALSE[,1,)
```

where x is the close price, nFast number of periods for fast moving average, nSlow number of periods for slow moving average and nSig number of periods for signal moving average, maType represents type of moving average applied and percent is a logical parameter, TRUE means absolute difference between the fast and slow moving average and FALSE means the percentage difference between the fast and slow mobbing difference. This function has two outputs, the first one is MACD, and the second one the signal, so number 1 at the end represents the first output, MACD.

The function `momentum`() and `ROC`() are similar, the ROC indicator provides the percentage difference of a series over two observations, while the momentum indicator simply provides the difference.

- `momentum(x, n = 1, ...)`
- `ROC(x, n = 1, type = c("continuous", "discrete"), ...)`

where type is compounding type either "continuous" (the default) or "discrete". So to achieve ROC from Table 3.1, we have to set `type` to "`discrete`".

Another price indicator is Stochastic Oscillator which measures where the close is in relation to the recent trading range. The function used is `stoch`():

```
stoch(HLC, nFastK = 14, nFastD = 3, nSlowD = 3, maType,
bounded = TRUE, smooth = 1, ...)
```

where HLC means High-Low-Close prices, nFastK is Number of periods for fast %K, nFastD is Number of periods for fast %D. This function has two outputs, the first one is fast %K, and the second one is the fast %D.

The syntax of remaining functions is self-explanatory and mostly default values of parameters can be used to obtain the required indicator so we omit a more detiled description.

In data preparation section, we use function `sample`() takes a sample of the specified size from the elements of data using either with or without replacement. And function `set.seed`() set the seed of R's random number generator, which is useful for creating simulations or random objects that can be reproduced. In the application, we use `set.seed`(7 or 16) to select the specific data set and make the whole process reproducible.

3.2.2 Growing Phase

The Recursive Partitioning and Regression Trees (rpart) programs build classification models of a very general structure, the resulting models can be represented as binary trees. There are some restrictions in rpart package concerning depth of the maximal tree.

```
rpart(formula, data, weights, subset, na.action = na.rpart,
method, model = FALSE, x = FALSE, y = TRUE, parms, control,
cost, ...)
```

- `formula`: It represents our model, as we want to classify Y according to X_1 ,…, X_n. We use formula = Y~ X_1 +…+X_n , or simply formula=Y~. If data contains only predictor Y and independent variables X_1,…, X_n.
- `data`: an optional data frame in which to interpret the variables named in the formula. Such as the data from training data set.
- `method`: The splitting rule depends on a type of a tree model. We can build trees based upon "anova" (regression), "poisson" or "exp" approach. If method is missing then the routine tries to make an intelligent guess. If Y is a factor then method = "class" is assumed, otherwise method = "anova" is assumed. We solve

classification problem in this thesis so Y as to be a factor and `method = "class"`.

- `control:` the list of options that control detail of the `rpart` algorithm trhough `rpart.control()`

```
rpart.control(minsplit=20, minbucket= round(minsplit/3), cp = 0.01,maxcompete = 4, maxsurrogate = 5, usesurrogate = 2, xval = 10, surrogatestyle = 0, maxdepth = 30, ...)
```

  - ➢ `minsplit`: The minimum number of observations in a node for which the routine will even try to compute a split. And the minimum number of observations that must exist in a node in order for a split to be attempted. That is, if we want to build a maximal tree we should set minisplit= 0.
  - ➢ `cp`: It already controls the pruning phase. As mentioned above, so we have to set cp =0 to obtain the maximal tree.
  - ➢ `xval`: It can control the number of cross-validations. In the thesis, we use 10-fold cross validation, so we set xval=10.
  - ➢ `maxdepth:` There are some limits for depth of a maximal tree, actually depth of at most 30 is possible. This parameter is helpful if we want to build a tree with given depth. For example, if we want to see only first split(s).

3.2.3 Pruning Phase and Tree Object attributes

When a tree object is obtained, typically the maximal tree (if possible), we use the following functions in pruning phase:

- `prune`: It allows us to generate a subtree for given value of cp.
- `printcp`: Displays the cp table for fitted rpart object. Using this command, we would get table which summarize the pruning phase graphs with data generated by V-fold cross-validation function.
- `plotcp`: It provides a graphical representation to the cross validated error summary which is output of printcp function. The cp values are plotted against the geometric mean to depict the deviation until the minimum value is reached.

From a rpart object, we should know how we can extract and what it included. A tree object is really complex. Here are some tips how we can extract some interesting characteristics:

- `maxtree$cptable`: It shows the complexity parameter and error of a sequence of subtrees, which from root tree to maximal tree.
- `maxtree$variable.importance`: It calculates the different importance of selected variables.
- `maxtree$frame`: It included all information of the maximal tree, such as the number of splits, terminal nodes and depth of the tree.

To draw trees, we use package `rattle`(Williams,2011) which present an intuitive point and click interface for data mining. The function `fancyRpartPlot`() Plots a fancy rpart decision tree using the pretty rpart plotter.

3.2.4 Prediction and Performance Evaluation

Before we get the confusion matrix, we should use function `predict.rpart`() from package `rpart` for predictions from the results of various model fitting functions.

```
predict(object,newdata,type=c("vector","prob","class","matrix"),)
```

It is used to predict values of target variable from given data set. In the command, we only write the `object` (decision tree in this thesis) and the `newdata` represents the given data set. The type "`prob`" gives us all possible classes together with probabilities of each class for given case (e.g. UP with 0.8 and DOWN with 0.2) and "`class`" gives us only the class with the highest probability. (eg. UP or DOWN)

Then we apply function `confusionMatrix`() from package `caret` (Kuhn,2020) to calculate a cross-tabulation of observed and predicted classes with associated statistics.

```
confusionMatrix(data, reference, positive,...)
```

In this thesis, data are predicted values of target variable for given data set calculated with prediction() function above and reference are actual values of target

variable in the data set. Parameter `positive` is as an optional character string for the factor level that corresponds to a "positive" result, eg. `positive = "UP"`.

If CM (confusion matrix) is object obtained with confusionMatrix we can get many information:

- `CM$table`: It is a 2x2 matrix table that categorizes predictions according to they match the actual value in the data. That is, the base for the confusion matrix Table 2.1 in Chapter 2.
- `CM$overall`: It shows the overall accuracy rate and computed along with a 95 percent confidence interval for this rate (using binom.test) and a one-sided test to see if the accuracy is better than the "no information rate," which is taken to be the largest class percentage in the data. It also included unweighted Kappa statistic and p-value from McNemar's test.
- `CM$byClass`: It contains the sensitivity, specificity, positive predictive value, negative predictive value and other characteristics introduced in section 2.2.3.1.

4 Stock Price Movement Prediction for Selected Company

Stock market prediction is considered to be a challenging task for both investors and researchers, due to its profitability and complexity. Highly accurate stock market predictive models are very often the basis for the construction of algorithms used in automated trading. In this thesis, predictive models are built using the CART algorithm. The models are built on the historical data of the Baidu, Inc. stock price. Several technical indicators, popular in quantitative analysis of stock markets, are selected as model inputs. The proposed method is empirically evaluated using 10-fold cross-validation, achieving a good classification accuracy.

In this chapter, we will use the method we described to analysis the data sets and show the result of prediction and performance evaluation.

4.1 Company Profile

Baidu, Inc. (BIDU) is a Chinese multinational technology company specializing in Internet-related services and products and artificial intelligence (AI), headquartered in Beijing and was founded in 1/1/2000. It is one of the largest AI and internet companies in the world. Baidu offers various services, including a Chinese search engine, as well as a mapping service called Baidu Maps. Baidu offers about 57 search and community services, such as Baidu Baike (an online encyclopedia) and a keyword-based discussion forum. Baidu has the second largest search engine in the world and held a 76.05% market share in China's search engine market. In December 2007, Baidu became the first Chinese company to be included in the NASDAQ-100 index.

4.2 Data Collection and Preparation

For the purpose of our analysis, we downloaded daily data from Yahoo! Finance (source: https://finance.yahoo.com/) and collected 3020 different observations from 1/1/2008 to 31/12/2019. For each trading day The 6 input variables were recorded, which included open, high, low, close and adjusted price and volume.

Let us download the data (daily volumes and open, high, low, close, and adjusted prices) with function `getSymbols()` from package `quantmod` which must first be loaded:

```
library(quantmod)
getSymbols("BIDU", from="2008-01-01",to="2019-12-31")
```

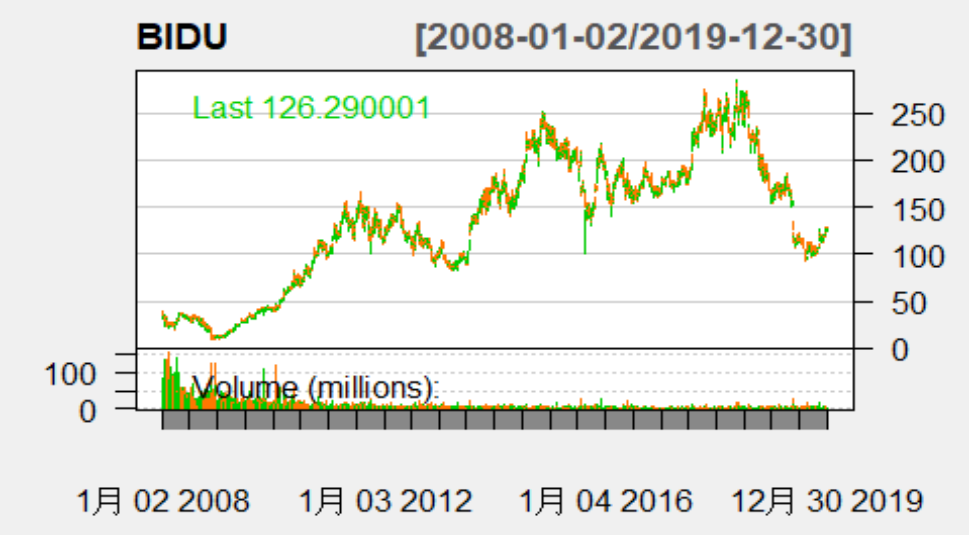
The first column name of the xts object BIDU, representing our time series, is "BIDU.open", the second "BIDU.high" etc. Because we analyze only one company, let simply change the column names to "open", "high", etc. Here are first 6 rows of data set in Table 4.1.

Table 4.1 Baidu data sets (first 6 observations)

Period	Open	High	Low	Close	Volume	Adjusted
2008/01/02	39.45	39.77	37.45	38.19	54820000	38.19
2008/01/03	38.34	38.36	36.78	37.51	47186000	37.51
2008/01/04	36.46	37.15	35.8	36.1	59227000	36.1
2008/01/07	35.92	36.4	33.5	34.43	71881000	34.43
2008/01/08	34.58	35.97	34.15	34.52	56220000	34.52
2008/01/09	34.26	34.78	31.92	34.41	83399000	34.41

We also show the closing price for a long period in Figure 4.1:

Figure 4.1: Historical closing price of BIDU



We will discuss 4 times windows (1, 5, 30, 90). The definition of our output variable is affected by analyzed time window. We show the whole process in detail for one specific time window of five days in Section 4.3 and 4.4 and so we introduce definition of target variable only for this specific window here.

4.2.1 Target Variable

Target variable (output, dependent variable) definition: We only want to know whether the close price increased or decreased. That is, we downloaded close prices Cl_1, Cl_2, … , Cl_{3019},Cl_{3020} , and for a five-day time window we calculate 3015 differences Cl_6-Cl_1, … Cl_{3020}-Cl_{3015}:

```
window <- 5
BIDU$change <- (lag(BIDU,k = -window)-BIDU)$Close
```

The change price of last five days are NA in Table 4.2 which we should delete them.

Table 4.2: Baidu data sets (last 6 observations)

Period	Open	High	Low	Close	Volume	Adjusted	change
2019/12/20	128	129.2	126.4	127.5	5101500	127.5	-1.999
2019/12/23	127.8	129.4	127.6	128.8	2440300	128.8	NA
2019/12/24	129	129.4	127.8	128.5	889400	128.5	NA
2019/12/26	128.6	128.9	127.2	127.6	1628500	127.6	NA
2019/12/27	128	128	125.9	126.4	2167100	126.4	NA
2019/12/30	125.9	127.9	125.3	126.3	2216700	126.3	NA

For 90-day window change, we calculate changes firstly and we cannot use 90 last cases. So we decided to Use only 2930 cases for all time windows to simplify the later comparison.

```
BIDU <- BIDU[-c((nrow(BIDU)-89):nrow(BIDU))]
```

We can start with definition of an R object we will use for or model building. We call it `treedata` and we will use `data.table` package for its definition. The first column is our target variable `Y.Close` with exactly two classes: UP, if the price increased, and DOWN if it decreased or did not change. (In order for simplicity, no change is included in DOWN.)

```
library(data.table)

Y.Close <- as.factor(ifelse(BIDU$change>0, "UP",  "DOWN"))

treedata <- data.table(Y.Close)
```

In Table 4.3, the frequency table of UP or DOWN movements for two-year periods are shown. As we can see, the classes are balanced which is good for CART algorithm to work properly:

Table 4.3: Frequency table

	DOWN	UP
2008/2009	240	265
2010/2011	196	308
2012/2013	232	270
2014/2015	254	250
2016/2017	229	274
2018/2019	225	187
Sum	1376	1554

Total probability of decrease in five days was 46.96% = 1376/2930, and increase was 53.04%=1554/2930.

4.2.2 Feature Extraction

Technical indicators provide insights to the expected stock price behavior in future. These technical indicators are used as features to train the classifiers. Our goal of thesis is to predict the stock price trend, so we choose these technical indicators to analysis the historical and future stock price movements.

These different technical indicators were selected for model input, and formulas for calculating the selected indicators are described in Table 3.1. First three indicators are moving averages. They are used to smooth the stock price signal, making it easier for an investor to identify trends. Another indicator, MACD, is calculated by subtracting the slow exponential moving average from the fast exponential moving average, and it is commonly used in combination with the technical indicators mentioned above to suggest buying or selling of specific stocks. RSI is another indicator, used to detect whether a stock is overbought or oversold. Its value ranges from 0 to 100. If the RSI value is above 70, this indicates that the stock is overbought, and if the RSI value is below 30, this indicates that the stock is oversold. The choice of Technical indicators as well as length of time windows was inspired by work of Manojlović and Štajduhar (2015)

For example, we calculate SMA for parameters that included 5, 10, 20 days. We use function `SMA()` from package `TTR` and add new columns to object `treedata`.

```
library(TTR)
treedata$sma5 <-SMA(Cl(BIDU),n=5)
treedata$sma10 <-SMA(Cl(BIDU),n=10)
treedata$sma20 <-SMA(Cl(BIDU),n=20)
```

- Weighted Moving Average

```
treedata$wma5 <-WMA(Cl(BIDU),n=5)
treedata$wma10 <-WMA(Cl(BIDU),n=10)
treedata$wma20 <-WMA(Cl(BIDU),n=20)
```

- Exponential Moving Average

```
treedata$ema5 <-EMA(Cl(BIDU),n=5)
treedata$ema10 <-EMA(Cl(BIDU),n=10)
treedata$ema20 <-EMA(Cl(BIDU),n=20)
```

- MACD

```
treedata$macd <-MACD(Cl(BIDU), nFast = 12, nSlow = 26,nSig=4, ma
Type=EMA, percent = FALSE)[,1]
```

- Momentum

```
treedata$Mom5<-momentum(Cl(BIDU),n = 5)
treedata$Mom10 <-momentum(Cl(BIDU),n = 10)
treedata$Mom20 <-momentum(Cl(BIDU),n = 20)
```

- ROC

```
treedata$Roc5 <-ROC(Cl(BIDU),n = 5,type = "discrete")
treedata$Roc10 <-ROC(Cl(BIDU),n = 10,type = "discrete")
treedata$Roc20 <-ROC(Cl(BIDU),n = 20,type = "discrete")
```

- RSI

```
treedata$rsi <-RSI(Cl(BIDU), n=14)
```

- The Williams Accumulation / Distribution (AD)

```
treedata$wad <- williamsAD(BIDU[,c("High","Low","Close")])
```

- Commodity channel index (CCI)

```
treedata$cci <- CCI(BIDU[,c("High","Low","Close")])
```

- On Balance Volume (OBV)

```
treedata$obv <- OBV(BIDU$Close,BIDU$Volume)
```

- Stochastic

```
treedata$Kprc <- stoch(BIDU[,c("High","Low","Close")],nFastK= 1
4, nFastD = 3)[,1]
treedata$Dprc <- stoch(BIDU[,c("High","Low","Close")],nFastK= 1
4, nFastD = 3)[,2]
treedata$Wprc <- WPR(BIDU[,c("High","Low","Close")])
```

Object treedata have 2930 rows and 24 columns. Because the long EMA is based on 26 days period, to avoid NA values and make the whole process more transparent we omit the first 25 cases and decrease number of rows in treedata to 2905.

```
treedata <- treedata[-(1:25)]
```

There are 6 cases and 23 different indicators can be found in Annex 1. As mentioned in section 2.2, there is no need to normalize or scale our input data for the CART algorithm. So we can start with training and testing data selection.

4.2.3 Training and Testing Data Sets

The model is building with training data set (e.g. 70%), we use the rest as testing data set (30%) later to evaluate the model performance.

We use `set.seed()` function to select the specific data set and make the whole process reproducible. Let us split the original data set:

```
set.seed(7)
rowstraining <- sample(x = 1:nrow(treedata),size = round(0.70*n
row(treedata)))
training <- treedata[rowstraining,]
testing <- treedata[-rowstraining,]
```

Here are the proportions of UP and DOWN values in training data set which the sum is 2033 and testing data set summed 872 respectively. The root error of training and testing data is 0.459=934/2033 and 0.483=421/872 respectively. (see Table 4.4)

Table 4.4: Training and testing data sets

	Training	Testing
DOWN	934	421
UP	1099	451
SUM	2033	872
Root Error	0.459	0.483

4.3 Building Model for a Five-day Window

4.3.1 Growing Phase

Let us use training data set for growing phase which result is maximal tree, the goal is to minimize empirical risk $R(T)$ and to achieve 100% accuracy for our training data.

```
library(rpart)
set.seed(16)
maxtree <- rpart(Y.Close~., data=training, control=rpart.contro
l(minsplit = 0,cp=0), method ="class")
```

Before we introduce the maximal tree generated by `rpart` package, let us describe how the first splits are generated to better understand the idea of recursive partitioning method. We can generate the one-split tree setting `maxdepth` in `rpart.control` to 1:

```
tree.first.split <- rpart(Y.Close~., data=training,
        control=rpart.control(maxdepth = 1), method ="class")
```

Object `tree.first.split` is a list containing information on the one-split tree. Among other things, the basic data on the first split:

```
> tree.first.split
n= 2033
node), split, n, loss, yval, (yprob)
      * denotes terminal node
1) root 2033 934 UP (0.4594196 0.5405804)
  2) wma10>=255.6609 40   4 DOWN (0.9000000 0.1000000) *
  3) wma10< 255.6609 1993 898 UP (0.4505770 0.5494230) *
```

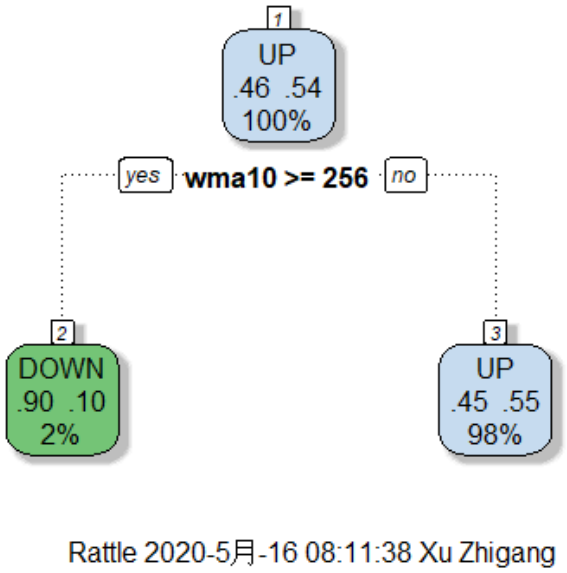
We can see 3 nodes, the root node, Node 1), and two terminal nodes, Node 2), and Node 3). Every terminal node in output is denoted by star (*). There are 2033 cases in Node 1), 45.59% belongs to DOWN class and 54.06% to UP class. Because proportion of UP class is higher than DOWN class, class UP was assigned to Node 1) and so 934 cases were incorrectly classified. The splitting rule is wma10>=255.6609 because this split has maximizing $\Delta(A, A_L, A_R)$, see equation (2.32), which called the primary split or the best split.

There are 40 cases in Node 2), 90% belongs to DOWN class and 10% to UP class. Because proportion of DOWN class is higher than UP class, class DOWN was assigned to Node 2) and so 4 cases were incorrectly classified.

We plot this decision tree using function `fancyRpartPlot` from package `rattle` You can see the plot in in Figure 4.2.

```
library(rattle)
fancyRpartPlot(tree.first.split)
```

Figure 4.2: First split tree



Let me denote the root node by A_1 and use the following notation (similar notation will be applied for all the nodes):

- n_{A1} is the number of observations in node A_1.
- $p(A_1) = 1 = 100\%$, probability of A_1 (for future observations); this node includes 100% of the data.
- $(\text{UP}|A_1) = 0.541 = 1099/2203$, probability of UP trend in node A_1 (for future observation).
- $(\text{DOWN}|A_1) = 0.459 = 934/2033$, probability of DOWN trend in node A_1 (for future observations).

Accuracy of the root node is 0.541 and the misclassification rate $R(T_{root})$ is 0.459. And accuracy of the maximal tree is 100%, and $R(T_{max})$ is 0%.

If a node is a pure node, then its risk is zero. When I look for a rule to split the node I want its sons to be as pure as possible. I apply Gini impurity function from Formula (2.31) in the following form to quantify purity of a node:

$$(A) = p(\text{UP}\,|A) \cdot (1 - p(\text{UP}\,|A)) + p(\text{DOWN}\,|A) \cdot (1 - p(\text{DOWN}\,|A))$$

$$= 1 - (\text{UP}\,|A)^2 - p(\text{DOWN}\,|A)^2.$$

That is,

$$(A_1) = 1 - p(\text{UP}\,|A_1)^2 - p(\text{DOWN}\,|A_1)^2 = 1 - (0.541)^2 - (0.459)^2 = 0.497.$$

There are a huge number of candidates for the best split A_L, A_R. We extracted 23 features, $X_1, X_2, \ldots, X_{23}$, all of them continuous. So all possible rules take form of $X_i \geq a_i$, i=1, …, 23, $a_i \in$ R. So to find the one maximizing $\Delta(A, A_L, A_R)$ is definitely a job for computer. Applying the rule which we can see in Figure 4.2, wma10>=255.6609, we obtain nodes Node 2) and Node 3) resulting in the following impurity

$$0.02(1 - 0.900^2 - 0.100^2) + 0.98(1 - 0.451^2 - 0.549^2) = 0.489 < 0.497$$

If we set other different rule for the split, we cannot get the results of impurity is lower than the root tree.

4.3.2 Maximal Tree

We generate new splits in the same way as we describe and the result is maximal tree and in Table 4.5, you can see the variable importance of maximal tree. We can see that mainly simple and weighted averages of all lengths play important role in the maximal tree building.

Table 4.5: Variable importance of T_{max}

sma5	sma10	wma10	sma20	wma20	wma5	rsi	ema10	obv	macd
252.3	216	188.3	175.1	162.3	160.4	152.3	146	145.3	144.6
wad	Mom5	Kprc	cci	ema5	Mom20	Wprc	Roc5	Mom10	Dprc
133.5	133.2	132.9	127.5	125.8	124.5	118.2	114.6	112.4	108.9
Roc20	ema20	Roc10							
105.1	100.6	100							

There are 741 nodes in the maximal tree and nsplit represents the number of splits. It is 370 in the Table 4.6 when real error is 0. The height of this tree is 25. It represents the maximal number of criteria for classifying the price movement.

4.3.3 Pruning Phase

We look for subtrees of the maximal tree (we prune the tree). The result is a sequence of subtrees, from root tree to maximal tree. Summary for found sequence can be printed with function `printcp(`T_{max}`)`. You can see the result in Table 4.6.

Table 4.6: cp table generated by printcp

	cp	nsplit	rel error	xerror	xstd
1	0.03426124	0	1.000000	1.000000	0.024058
2	0.01713062	1	0.965739	0.978590	0.024014
3	0.01124197	7	0.831906	0.960390	0.023970
*					
16	0.00214133	104	0.316916	0.804070	0.023300
17	0.00178444	153	0.211991	0.783730	0.023173
18	**0.00160600**	**157**	**0.204497**	**0.768740**	**0.023073**
19	0.00142755	175	0.175589	0.773020	0.023102
20	0.00107066	181	0.167024	0.774090	0.023109
21	0.00080300	300	0.039615	0.771950	0.023095
22	0.00053533	308	0.033191	0.775160	0.023117
23	0.00000000	370	0	0.77088	0.023088

If we multiply this output with root node error $R(T_{root})$, we can see real misclassification errors, the summary can be seen in Table 4.7.

Table 4.7: Adjusted cp table (in terms of α)

	α	|T|	$R(T)$	$R_{CV}(\beta)$	$SE_{CV}(\beta)$
1	0.0157402853	1	0.45941958	0.4594196	0.01105264
2	0.0078701426	2	0.44367929	0.4495819	0.01103271
3	0.0051647811	8	0.38219380	0.4412199	0.01101233
*					
16	0.0009837678	105	0.14559764	0.3694048	0.01070429
17	0.0008198065	154	0.09739302	0.3600590	0.01064604
18	**0.0007378259**	**158**	**0.09394983**	**0.3531727**	**0.01060032**
19	0.0006558452	176	0.08066896	0.3551402	0.01061363
20	0.0004918839	182	0.07673389	0.3556321	0.01061692
21	0.0003689129	301	0.01819970	0.3546483	0.01061032
22	0.0002459420	309	0.01524840	0.3561240	0.01062021
23	0.0000000000	371	0	0.3541564	0.01060700

In Table 4.6, the first column is the order of a tree. The second column is the cp parameter used in function `rpart` introduced in Chapter 3. This two tables can be the whole in Annex 3&4. When we multiply it by root node error which is 0.459 = 934/2033, we can get the complexity parameter α. The third column is number of the splits which means the splits of the trees. The fourth, fifth and sixth column multiplied by root node error are a misclassification rate from the learning data set, mean misclassification rate generated by cross validation $R_{CV}(\beta)$ and its standard error $SE_{CV}(\beta)$.

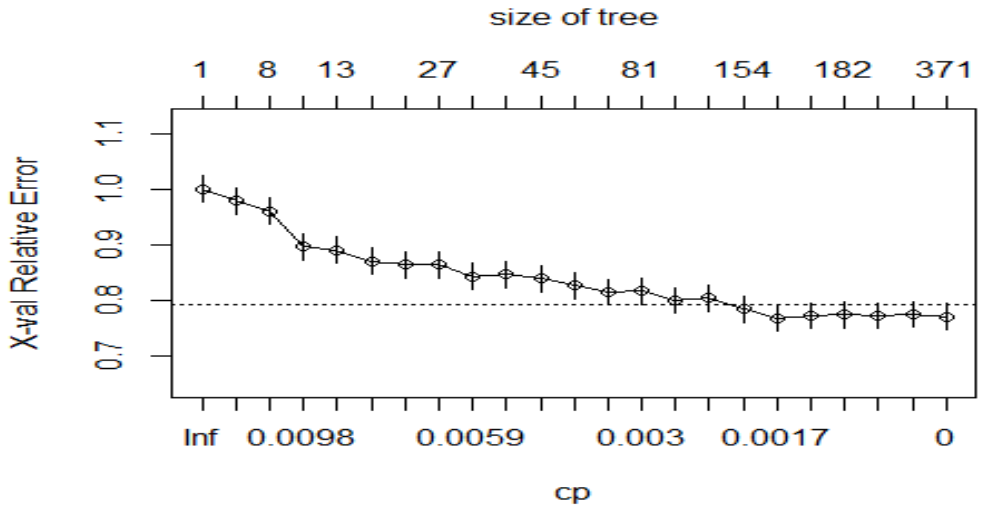
The first row represents our root node. As we mentioned above when we set cp to any value from the interval [0.03426124, ∞) then we generate the tree without any splits. In terms of complexity parameter $\alpha \in [0.0157402853, \infty)$.

In the second row, when we set $cp \in [0.01713062, 0.03426124)$, in terms of complexity parameter $\alpha \in [0.0078701426, 0.0157402853)$. I can get the tree with only one split.

Similarly, in the 23th row, when we set $cp \in [0, 0.00053533)$, in terms of complexity parameter $\alpha \in [0, 0.000245942)$. We can simply set α=0 to get the maximal tree with 371 terminal nodes.

A graphical representation of sequence of cps are from Table 4.6 and it can be seen in the lower horizontal axis in Figure 4.3. The higher horizontal is the size of trees which is the |T| in Table 4.7. And the vertical axis represents X-val Relative Error.

Figure 4.3: plotcp of T_{max}



4.3.4 10-Fold Cross-Validation and Optimal Tree

As I described in Chapter 2, I chosen 10 subsets randomly. The choice depends on number 16 in `set.seed( )` preceding definition of T_{max} introduced in growing phase in Section 4.3.1. Nine of them are training data sets, the other one is testing data set.

The simplest method to choose a good model (tree) is to choose the tree with minimal estimate of true misclassification rate (with minimal cross-validation error). Each subtree is connected with its complexity parameter α. cp in printcp output is proportional to α, so we can find the cp (0.00160600) for tree with minimal xerror this way:

```
mincp <- maxtree$cptable[which.min(maxtree$cptable[,'xerror']),
'CP']
```

Following notation from Chapter 2 we are looking for β_{min} with the smallest cross-validation error $R_{CV}(\beta_{min})$. If we combine the Table 4.6 and plotcp output of T_{max} (Figure 4.3), we can choose the 18th as the minimal error tree.

That is, when we take $\alpha \in [0.0007378259, 0.0008198065)$. We can generate the tree with 158 terminal nodes. As I described in the Chapter 2, we have to choose a representative from the interval, namely, square root of a product of the limits:

$$\beta_{18} = \sqrt{\alpha_{17} \cdot \alpha_{18}} = \sqrt{0.0007378259 \cdot 0.0008198065} = 0.000777737$$

Applying Formula (2.37) in Chapter 2:

$$R_{CV}(\beta_{18}) = \frac{1}{10}\sum_{i=1}^{10} R\left(T_{18}^{(i)}\right) = 0.768740 * (\frac{934}{2203}) = 0.3531727,$$

which is $R_{CV}(\beta)$ shown in Table 4.7 from Formula (2.38), and

$$SE_{CV}(\beta_{18}) = \sqrt{\frac{R_{CV}(\beta_{18})(1 - R_{CV}(\beta_{18}))}{N}} = \sqrt{\frac{0.3531727(1 - 0.3531727)}{2033}} = 0.01060032,$$

which is $SE_{CV}(\beta)$ shown in Table 4.7.

The variable importance of $T(\beta_{18})$ in Table 4.8. We can see that mainly simple and weighted averages play important role in this subtree building and RSI , OBV become more important.

Table 4.8: Variable importance of $T(\beta_{18})$

sma10	wma10	sma5	ema10	wma20	rsi	obv	wad	sma20	wma5
158.6	145.6	143.9	134.8	124.2	123	121	117.2	114.1	114
ema5	cci	Kprc	Mom5	Wprc	macd	Mom20	Roc5	Mom10	Dprc
109.1	104	100.6	95.47	95.02	94.64	93.93	90.75	90.32	89.07
ema20	Roc20	Roc10							
86.56	81.47	78.97							

There are 315 nodes in $T(\beta_{18})$, the tree with minimal cross-validation error. The nsplit of $T(\beta_{18})$ represents the number of splits which is 158. The height of this tree is 20. It represents the maximal number of criteria for classifying the price movement.

Another method to find the optimal model takes also standard deviation of those cross-validation errors into account. (See Section 2.2.2.2)

Note that choice of β_{min} can be affected by the seed of the random number generator to separate learning data set into 10 subtests. Small changes can be large changes in number of terminal nodes in optimal tree. To choose the simplest tree whose accuracy is comparable to $R_{CV}(\beta_{min})$, 1-SE rule is applied to select the right sized tree: An optimal tree, T (β_{opt}), is the tree corresponding to β_{opt} where β_{opt} is the maximum β satisfying Formula (2.40):

$$R_{CV}(\beta_{opt}) \leq R_{CV}(\beta_{18}) + SE_{CV}(\beta_{18}) = 0.36377302.$$

If we look at the $T(\beta_{16})$, $T(\beta_{17})$,

$$R_{CV}(\beta_{16}) = 0.3694048,$$

$$R_{CV}(\beta_{17}) = 0.3600590,$$

we obtain such a result for $R_{CV}(\beta_{17})$

$$R_{CV}(\beta_{17}) \leq R_{CV}(\beta_{min}) + SE_{CV}(\beta_{min}),$$

but for the $R_{CV}(\beta_{16})$

$$R_{CV}(\beta_{16}) \quad > \quad R_{CV}(\beta_{min}) + SE_{CV}(\beta_{min}).$$

We can find it in plotcp in Figure 4.3 and the 17th subtree is the first below the plotted line. That is, we can regard $T(\beta_{17})$ as the optimal tree with respect to 1-SE rule.

We can use R program to solve the problem in general. We find minimal cp value within 1-SE range (`mincp.1SE`) and use it as a complexity parameter in pruning phase to generate optimal tree $T(\beta_{17})$ (`opt.cv.tree`).

```
minerror <-
maxtree$cptable[which.min(maxtree$cptable[,'xerror']),'xstd']
mincp.1SE <-
maxtree$cptable[which.max(maxtree$cptable[,'xerror']<min(maxtr
ee$cptable[,'xerror'])+minerror),'CP']
opt.cv.tree <- prune(maxtree,mincp.1SE)
```

The variable importance of optimal tree in Table 4.9. We can see that the importance is very similar with $T(\beta_{18})$. The ROC still the least important for the model.

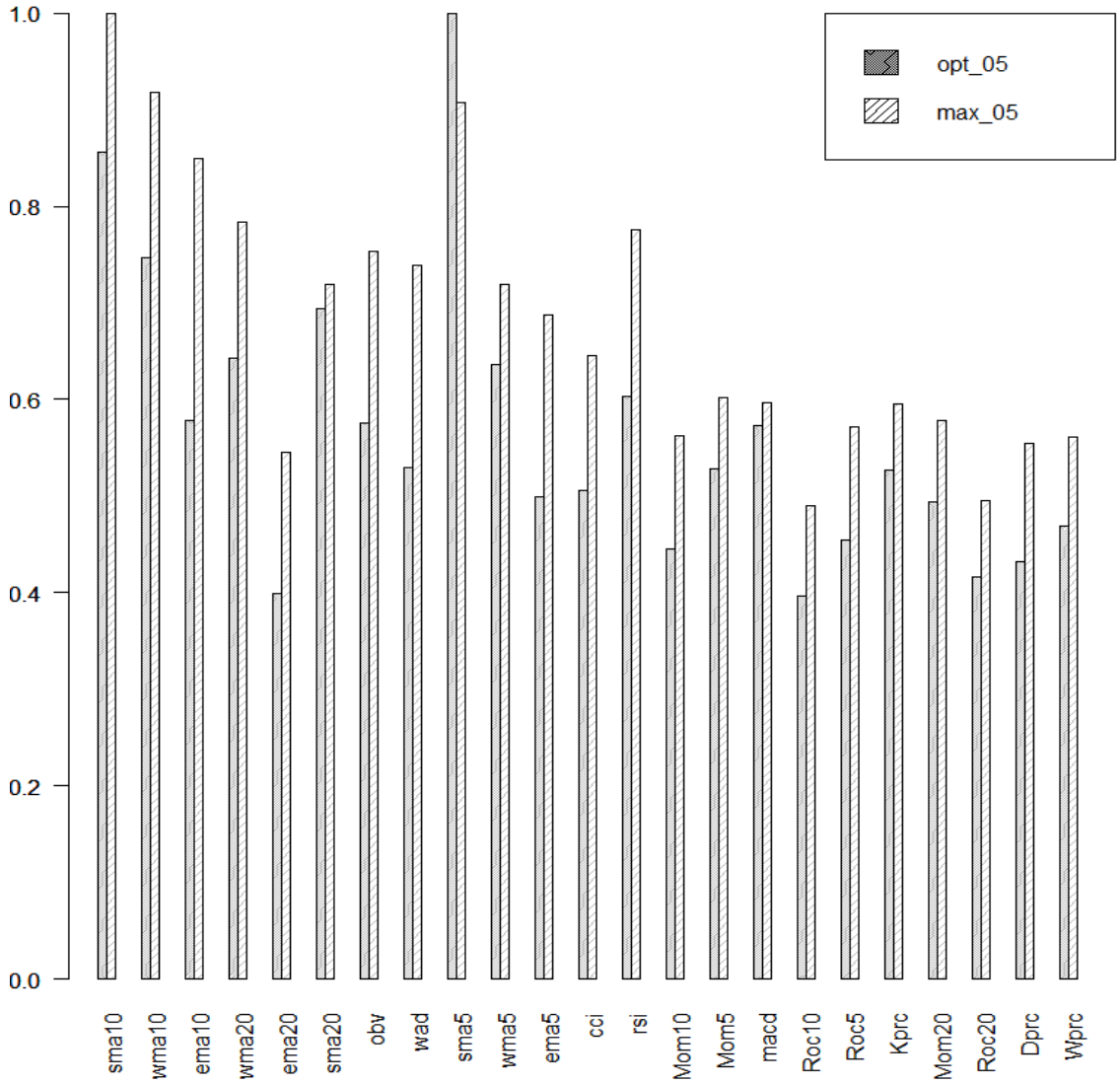
Table 4.9: Variable importance of $T(\beta_{17})$

sma10	wma10	sma5	ema10	wma20	rsi	obv	wad	sma20	wma5
158.6	145.6	143.9	134.8	124.2	123	119.4	117.2	114.1	114
ema5	cci	Mom5	macd	Kprc	Mom20	Roc5	Mom10	Wprc	Dprc
109.1	102.4	95.47	94.64	94.48	91.68	90.75	89.25	88.91	87.91
ema20	Roc20	Roc10							
86.56	78.48	77.72							

There are 307 nodes in the $T(\beta_{17})$. The number of splits is 154 and the height of this tree is 20. It represents the maximal number of criteria for classifying the price movement with this optimal tree.

In Figure 4.4, we generate the plot of variable importance for optimal (β_{17}) and maximal tree in five-day window. Values from Table 4.9 and Table 4.5 were divided by their maximal values, so number 1 was assigned to the feature with the highest impact in model building: sma10 for maximal tree and sma5 for optimal tree. As the optimal tree is nothing but a sub-tree of maximal tree we can see different patterns for the variable importance. E.g. if we want our model to fit the training data better the role of sma5 increases.

Figure 4.4: Variable importance for optimal and maximal trees in five-day window



4.4 Performance Evaluation for a Five-day Window

4.4.1 Prediction

We can use a developed model for prediction of price movements now. We set aside the testing data set to be able to evaluate performance of our predictive models. Consider one single day from the testing data set, for example the 138th row in Table 4.10 shows details.

Table 4.10: Features of No.138 day

Y.Close	sma5	sma10	sma20	wma5	wma10	wma20	ema5	ema10
UP	38.29	38.8	38.61	38.02	38.48	38.83	38.22	38.45
ema20	macd	Mom5	Mom10	Mom20	Roc5	Roc10	Roc20	ema20
37.99	0.8468	-1.416	-1.836	3.862	-0.03575	-0.0458	0.1125	37.99
rsi	wad	cci	obv	Kprc	Dprc	Wprc		
54.33	-12.44	-28.52	-217634000	0.3783	0.2106	0.6217		

We can use any of those three trees above to predict decrease or increase in stock price (in five days, UP is correct prediction). We use function `predict` to get the predicted class and the probability of prediction. We show some prediction results for the 138th row of testing data set.

- Prediction with maximal tree:

```
predict(object = maxtree,newdata = testing[138,],type='prob')
```

DOWN	UP
0	1

- Prediction with T(β_{18}):

```
predict(object = min.cv.tree,newdata =
testing[138,],type='prob')
```

DOWN	UP
0.02778	0.9722

- Prediction with T(β_{17}):

```
predict(object = opt.cv.tree,newdata =
testing[138,],type='prob')
```

DOWN	UP
0.1616	0.8384

The class UP was predicted with high probability of 83.84% in optimal tree $T(\beta_{17})$ and was 97.22% in $T(\beta_{18})$, even 100% in maximal tree. These three predictions were correct. The probability varies, but predicted class is always correct for this specific 138^{th} row.

We apply the prediction process to every case from testing data set and we can find number of correctly classified and misclassified cases in this set. It can help us to understand possible future predictions. We can see summary of received accuracies for five-day window in Table 4.11 together with accuracies resulting from application to training data described before.

Table 4.11: Accuracies of selected trees for training and testing data

	Training data	Testing data
T_0	54.06%	51.72%
$T(\beta_{17})$	90.26%	67.55%
$T(\beta_{18})$	90.60%	68.00%
T_{max}	100.00%	68.00%

As you can see, for training data, the accuracies of $T(\beta_{17})$, $T(\beta_{18})$ and T_{max} have to be much higher than testing data. For the T_{max} in training data is 100% but in testing data is only 68%. Accuracies for testing data are comparable to $1 - R_{CV}(\beta)$. For $T(\beta_{17})$ in testing data, as an example, we can expect $1 - R_{CV}(\beta_{17}) = 1 - 0.3601 = 63.99\%$ accuracy for this optimal tree and our testing data set yielded 67.55%, even more than estimated value, which is a positive result for us.

4.4.2 Confusion Matrix

Let us focus on 1-SE rule Optimal Tree. We can use function `confusionMatrix` from package `caret` to generate confusion matrix and even more information. The data is from testing data set and positive event is that close price is going UP:

```
library(caret)
confusionMatrix(data = predict(object = opt.cv.tree,newdata = t
esting,type = "class"),reference = testing$Y.Close,positive = "
UP")
```

Table 4.12: Optimal tree prediction of test data set

		Reference		
		DOWN	UP	Total
Prediction	DOWN	273	135	408
	UP	148	316	464
	Total	421	451	872

Table 4.12 includes cross-tabulation of predicted values and actual values. We can see that 273+316 =589 days were classified correctly. On the other hand, we have 135 days are UPs but we classify them as DOWNs and 148 days which are DOWNs but we classify them as UPs. The misclassification error of the optimal tree $T(\beta_{17})$ is

$$R(T(\beta_{17}))= \frac{135+148}{273+135+148+316} = 32.45\%$$

The accuracy of this classification is 100% - 32.45% = 67.55%. To summarize, this model is able to recognize 316 of 451 days UP from our testing data set, which is 70.07% (sensitivity), and is also able to recognize 273 of 421 days DOWN from our testing data set, which is 64.85% (specificity).

Table 4.13: Confusion matrix

Accuracy	0.6755
95% CI	(0.6433, 0.7065)
No Information Rate	0.5172
P-Value [Acc > NIR]	<2e-16
Kappa	0.3495
Mcnemar's Test P-Value	0.4756
Sensitivity	0.7007
Specificity	0.6485
Pos Pred Value	0.681
Neg Pred Value	0.6691
Prevalence	0.5172

A 95% confidence interval proportion of correctly classified cases is from 0.6433 to 0.7065. No information rate is similar with prevalence, which is 0.5172 = 451/872 means how often UP actually occurs in our testing data set. A one-sided test to make sure that the accuracy is better than the no information rate was performed yielding p-value less than $2 \cdot 10^{-16}$, so if we use significance level of 5% we fail to reject that the accuracy is better and it supports the model. Both confidence interval and test are using the exact Clopper-Pearson binomial test.

Kappa is account for how well the agreement between the model's predictions and the true values. It is 0.3495 in $T(\beta_{17})$ which presents fair agreement.

4.5 Predictive Models for 1, 30 and 90-day Windows

In this section, we briefly describe the other time windows and make a comparison among the four different days window with respect to the root error, the characteristics of trees and model performance measurement. There is an R code in Annex 9 for the five-day window but it can be used for any number of days between one and 90. Simply change number 5 in command `window <- 5` to any integer representing number of days you want to study.

Firstly, we should show root error of four time windows. (See Table 4.14.)

Table 4.14: Root error of Training data sets in four windows

	1	5	30	90
DOWN	1027	934	923	898
UP	1006	1099	1110	1135
SUM	2033	2033	2033	2033
Root Error	0.494835	0.459420	0.454009	0.441712

The UPs in training data set is increasing gradually from one-day to 90-day window. And the root error of data set is in downtrend.

Then from Table 4.15, we get general characteristics of maximal and optimal trees.

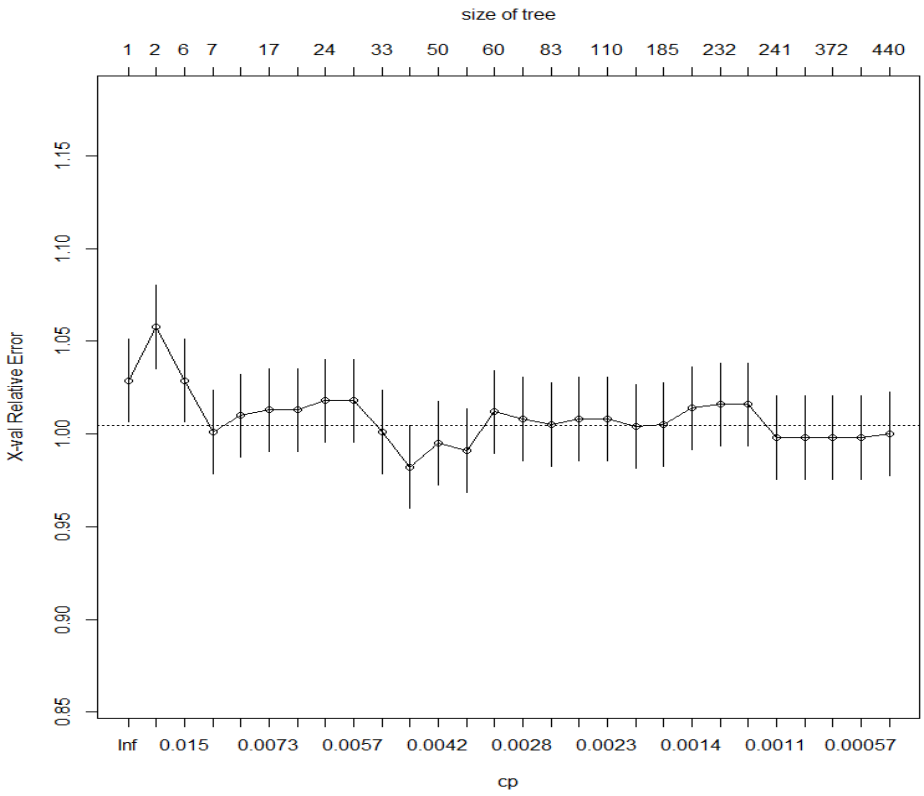
Table 4.15: Characteristics for maximal and optimal trees.

	Maximal trees			
Time window	1	5	30	90
Number of nodes	879	741	401	311
Number of terminal nodes	440	371	201	156
Depth	27	25	17	16
	Optimal trees			
Time window	1	5	30	90
Number of nodes	13	307	179	247
Number of terminal nodes	7	154	90	124
Depth	6	20	15	15

In maximal tree, we can see that the number of nodes, terminal nodes or the depth are decreasing by increasing time window which represents the tree become simpler.

We can see that depth of optimal tree for a one-day period decreased from 27 to 6. So the optimal tree is simple, but the reason is that we are not able to achieve significantly higher accuracy by adding more splits. See plotcp for one-day window **(**Figure 4.5**)** and 30, 90-day in Annex 6&8 that those two are similar to five-day (Figure 4.3). And adjusted cp table for one-, 30,90-day can be seen in Annex 2&5&7.

Figure 4.5: plotcp of T_{max} in one-day window



The dotted line is based on a 1-SE rule to get the upper limit, the first closest dot is T(β_4) which has 7 terminal nodes and cp is 0.007952, $R_{CV}(\beta_4)$ is 0.495327 which is xerror multiplied by root error (0.494835) see below Table 4.16.

Table 4.16: Optimal trees

	1	5	30	90
cp	0.007952	0.001784	0.001806	0.000557
Root Error	0.494835	0.459420	0.454009	0.441712
α	0.003935	0.000820	0.000820	0.000246
$R_{CV}(\beta)$	0.495327	0.360059	0.180030	0.112641

From Table 4.16, there is slight decrease in α which means the model become more complex by smaller α. And $R_{CV}(\beta)$ is dropping sharply by the increasing of window which means the lower expected error of the tree.

The performance measures for optimal trees is important criterion for evaluating the model.

Table 4.17: Performance measurement for optimal trees

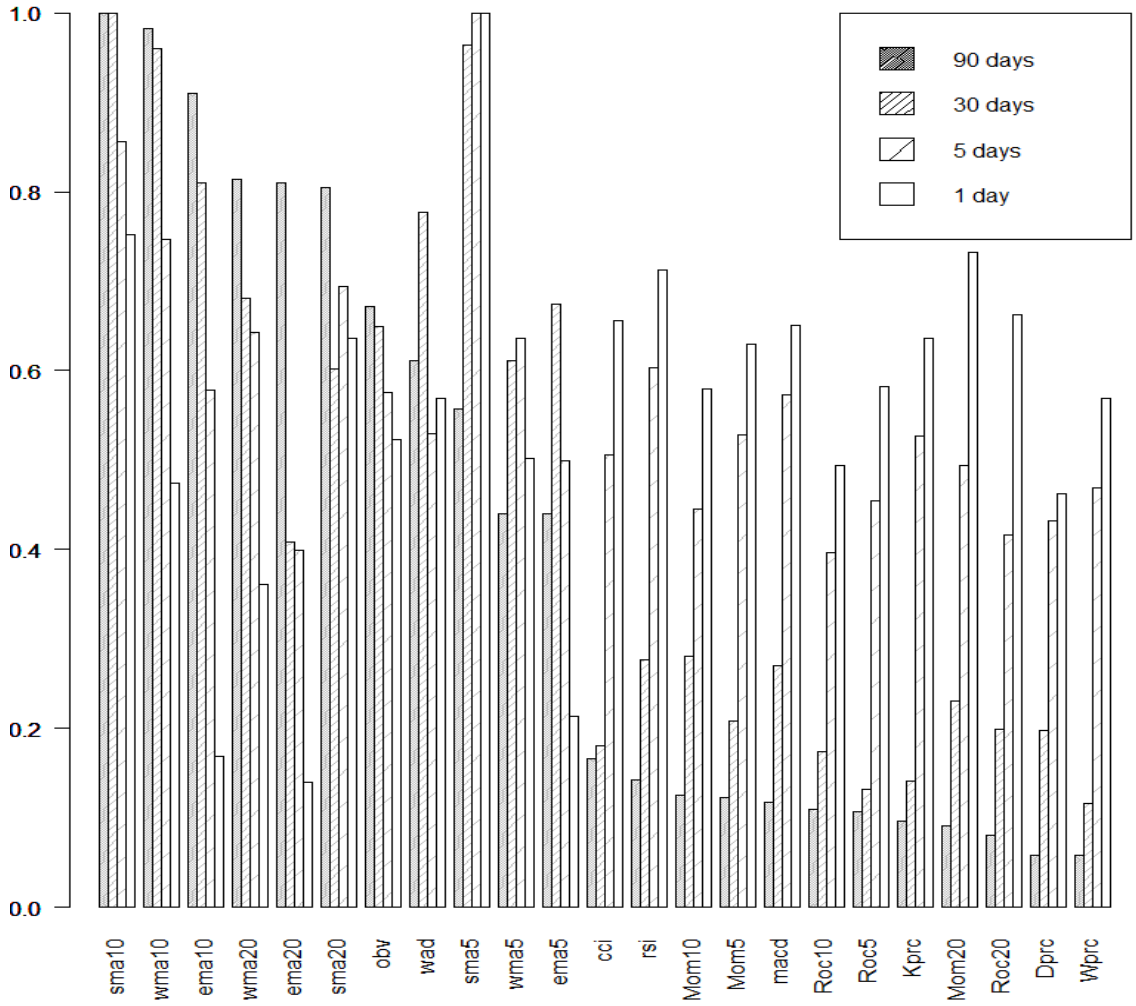
Window	1	5	30	90
Accuracy	0.5080	0.6755	0.8108	0.8876
Sensitivity	0.2706	0.7007	0.8217	0.9140
Specificity	0.7454	0.6485	0.7985	0.8575
PPV	0.5153	0.6810	0.8200	0.8799
NPV	0.5054	0.6691	0.8005	0.8972
kappa	0.0161	0.3495	0.6204	0.7736
F1-score	0.3549	0.6907	0.8209	0.8966
Prevalence	0.5000	0.5172	0.5275	0.5333

From the all ratios in Table 4.17, it represents the performance are much better from one-day window to 90-day window on the whole. The accuracy increases from 0.5080 to 0.8876. kappa is 0.0161 in one-day window which is in poor agreement and climbs rapidly to 0.7736 in good agreement. As for F1-score, it increased two time from 0.3549 to 0.8966 which means is the true positive rate and precision of the window have great improvement.

The order of features is given by 90-day window and is the same in both Figures, because optimal tree for 90 is almost maximal tree.

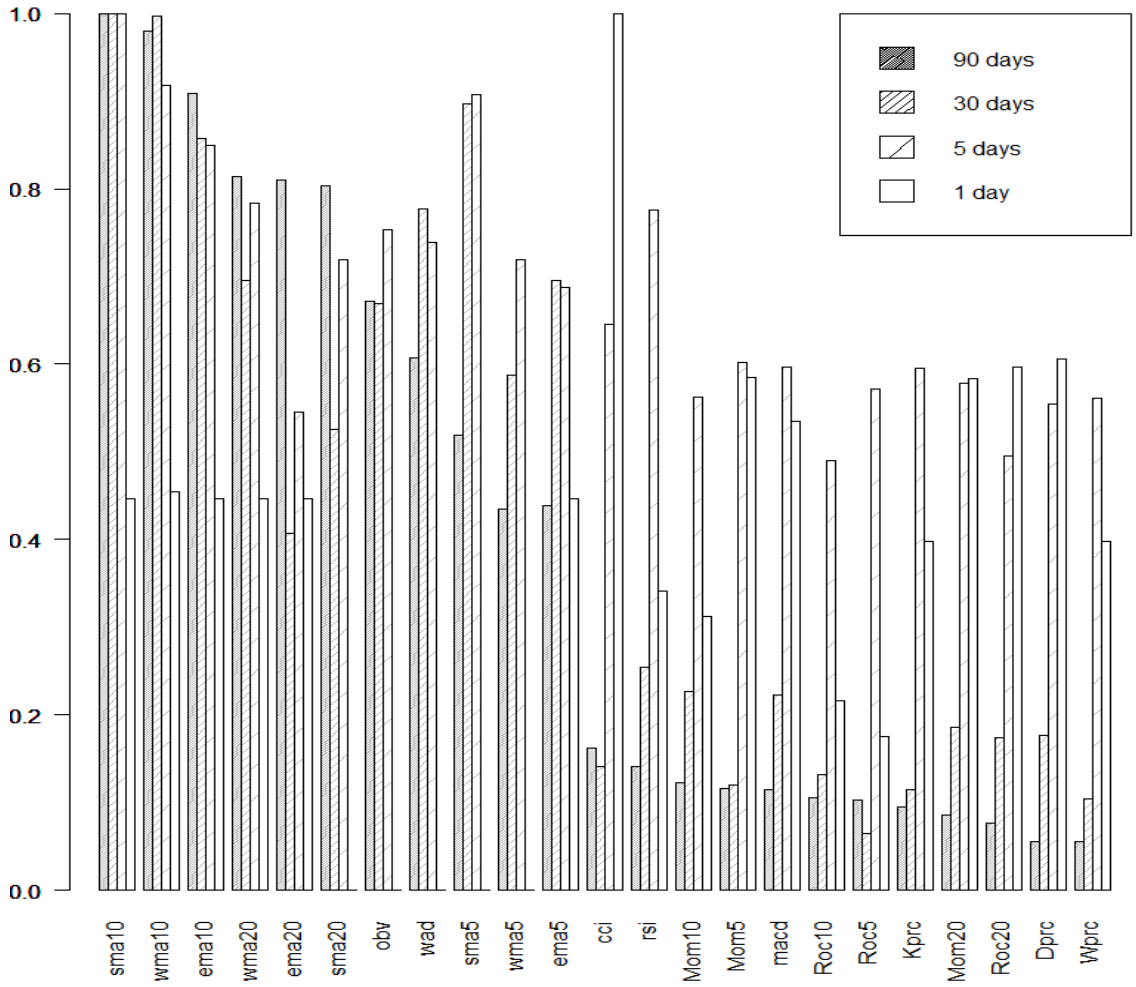
It seems that trend indicators like moving averages play more important role in long-term windows (30 and 90 days), but for short-term windows (five-day and one-day), we recognize price indicators play more significant role such as CCI, RSI, Momentum, ROC, Stochastic Oscillator. (See Figure 4.6)

Figure 4.6: Variable importance for maximal trees



For Optimal tree, the differences between long-term and short-term windows in price indicators are greater.

Figure 4.7: Variable importance for optimal trees



In Figure 4.7, if a certain variable of variables importance is shown as 0.0, for example, the obv or wad in one-day window, which means they have never split by the column. That is why their relative importance is 0.00 and their contribution to the model will be considered zero. The reason why variable importance is so different because the model is so pure that the model of one-day window has low accuracy.

5. Conclusion

In this thesis, we used the CART algorithm to build four models to be used to predict directions of Baidu's stock price for a one-day, five-day, 30-day, and 90-day time windows. Training data used to train the ML algorithm was the daily stock prices from the beginning of 2008 to the end of 2019 labeled with class UP if the price will increase during the studied period or with class DOWN if the price will decrease (or will not change) during the given period. Our results show that CART can be successfully utilized for building predictive models for predicting the direction of stock market trends for long-term periods.

The description of methods and algorithm in Chapter 2 and Chapter 3 aims to show how we use R program and R studio for this purpose. We used 70% of downloaded data and 10-fold cross-validation to find optimal models and remaining 30% of the data set was used for the final performance evaluation.

As a consequence, with the help of optimal models developed in Chapter 4, we can expect our prediction of the 30-day price movement to be correct with approximately 81.08% accuracy and with 88.76% for the 90-day price movement. On the other hand, the accuracy for the five-day period was only 67.55% and for the one-day window the model is not applicable because of accuracy of 50.80%.

To summarize, the root error is in downtrend from one-day window to 90-day window. The maximal tree of all time windows are gradually simpler. As for the performance measurement for optimal tree based upon training data, the accuracy increases from 50.80% to 88.76%. Those accuracies are comparable to estimates from 10-fold cross-validation ranging from 50.47% to 88.74%. It implies the longer window, the more accurate model. Secondly, we recognize that trend indicators like moving averages play more important role in long-term windows (30 and 90 days). Price indicators play more significant role for short-term windows (five-day and one-day), but even this shift could not maintain similar accuracy as long-term model have.

There are several research directions that should be pursued in the future. One of them involves improving the predictive accuracy of the model. We believe that this can be done in two ways. First, other technical indicators should be explored, especially the

indicators that include more volume. And fundamental variables can be considered. Next, different classification algorithms or ensemble methods like random forests can be used to further improve the model accuracy. We can also analyze receiver operation characteristics for another model evaluation and comparison. Consequently, the best obtained predictive models can be used for portfolio management.

Bibliography

Professional book

[1] ABHIJIT, Ghatak. *Machine Learning with R*. Springer Nature Singapore Pte Ltd.,2017. ISBN 978-981-10-6807-2

[2] BREIMAN, L., J. H. FRIEDMAN, R. A. OLSHEN and C. J. STONE. *Classification and regression trees.* New York: Chapman & Hall, 1993. ISBN 978-0-412-04841-8.

[3] SCOTT, V Burger. *Introduction to Machine Learning with R*. O'Reilly Media, Inc, 2018. ISBN 978-1-491-97644-9

[4] KIRKPATRICK, Charles D. and Julie R. DAHLQUIST. *Technical analysis: the complete resource for financial market technicians*. 3rd ed. Old Tappan: Pearson Education, 2016. ISBN 978-0-13-413704-9.

[5] LANTZ, Brett. *Machine learning with R: discover how to build machine learning algorithms, prepare data, and dig deep into data prediction techniques with R*. 2nd ed. Birmingham: Pack Publishing, 2015.Community experience distilled. ISBN 978-1-78439-390-8.

Article

[6] JENSEN, Michael C., Some Anomalous Evidence Regarding Market Efficiency (May 4, 1978). Journal of Financial Economics, Vol. 6, Nos. 2/3, pp. 95-101, 1978. Available at SSRN: https://ssrn.com/abstract=244159

[7] LANDIS, J. Richard, and Gary G. KOCH. *The Measurement of Observer Agreement for Categorical Data*. Biometrics. 1997, 33, 159-174.

[8] QUINLAN, J.R. *Induction of Decision Trees*. Kluwer Academic Publishers, Boston. Machine Learning.1986, 1: 81-106.

[9] MANOJLOVIĆ, Teo and Štajduhar, IVAN. *Predicting stock market trends using random forests: A sample of the Zagreb stock exchange.* In: 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). IEEE, 2015. p. 1189-1193.

[10] THERNEAU, T.M and ATKINSON, E.J.*An Introduction to Recursive Partitioning Using the RPART Routines*. Technical report, Mayo Foundation ,1997.

Electronic Bibliography and others

[11] RYAN A Jeffrey. and Ulrich M JOSHUA. (2020). *quantmod: Quantitative Financial Modelling Framework*. 2020, Available on https://CRAN.Rproject.org/package=quantmod

[12] ULRICH, Joshua. *TTR: Technical Trading Rules*. 2019, Available on https://CRAN.R-project.org/package=TTR

[13] KUHN, Max. *caret: Classification and Regression Training*. 2020, Available on https://CRAN.R-project.org/package=caret

[14] DOWL, MATT and Srinivasan, ARUN. *data.table: Extension of `data.frame`*. 2019, Available on https://CRAN.Rproject.org/package=data.ta

[15] R Core Team. *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. 2014, Available on http://www.R-project.org/

[16] THERNEAU, Terry, and Atkinson, BETH. *rpart: Recursive Partitioning and Regression Trees*. 2019, Available on https://CRAN.R-project.org/package=rpart

[17] WILLIAMS, G. J. *Data Mining with Rattle and R: The Art of Excavating Data for Knowledge Discovery, Use R!*. Springer. 2011.

List of abbreviations

α_i	Sequence of complexity parameters from pruning phase
β_i	Geometric means of complexity parameters for cross-validation
A_i	Node of a tree
ACC	Accuracy
CART	Classification and regression trees
CCI	Commodity Channel Index
cp	Proportional complexity parameter with respect to $R(T_0)$
EMA	Exponential Moving Average
EMH	Efficient Market Hypothesis
I(A)	Impurity of node A
MACD	Moving Average Convergence Divergence
NPV	Negative predictive value
OBV	On Balance Volume
PPV	Positive predictive value
$R_{CV}(\beta)$	Cross-validation error for T_β
R(Ai)	Risk of node Ai
R(T)	Risk of tree T
ROC	Rate of Change
RSI	Relative Strength Index
rpart	Recursive Partitioning and Regression Trees
$SE_{CV}(\beta)$	Standard error of $R_{CV}(\beta)$
SMA	Simple Moving Average
$\|T\|$	Number of leaf nodes in T
T_0	Root node
T_β	Pruned subtree w.r.t β
T_{max}	Maximal tree
T_{opt}	Optimal tree
TPR	True positive rate
TNR	True negative rate
TRH	True Range High
TRL	True Range Low
WMA	Weighted Moving Average
X_i	Independent variables
Y	Dependent variable

Declaration of Utilisation of Results from the Diploma Thesis

Herewith I declare that

- I am informed that Act No. 121/2000 Coll. – the Copyright Act, in particular, Section 35 Utilisation of the Work as a Part of Civil and Religious Ceremonies, as a Part of School Performances and the Utilisation of a School Work – and Section 60 – School Work, fully applies to my bachelor thesis;

- I take account of the VSB – Technical University of Ostrava (hereinafter as VSB-TUO) having the right to utilize the bachelor thesis (under Section 35(3)) unprofitably and for own use ;

- I agree that the bachelor thesis shall be archived in the electronic form in VSB-TUO's Central Library and one copy shall be kept by the supervisor of the bachelor thesis. I agree that the bibliographic information about the bachelor thesis shall be published in VSB-TUO's information system;

- It was agreed that, in case of VSB-TUO's interest, I shall enter into a license agreement with VSB-TUO, granting the authorization to utilize the work in the scope of Section 12(4) of the Copyright Act;

- It was agreed that I may utilize my work, the bachelor thesis or provide a license to utilize it only with the consent of VSB-TUO, which is entitled, in such a case, to claim an adequate contribution from me to cover the cost expended by VSB-TUO for producing the work (up to its real amount).

Ostrava dated 29.05.2020

徐志刚 Xu Zhigang

.....

Student's name and surname

List of Annexes

Annex 1: First 6 cases of Treedata

Y.Close	sma5	sma10	sma20	wma5	wma10	wma20	ema5	ema10
UP	25.15	26.59	28.34	24.37	25.59	26.96	24.73	26.15
UP	24.44	25.93	27.81	23.79	25.01	26.49	24.28	25.65
UP	23.93	25.48	27.4	23.86	24.78	26.19	24.41	25.47
UP	23.77	25.23	26.93	24.07	24.61	25.92	24.46	25.3
DOWN	24.39	25.25	26.64	24.85	24.77	25.84	25.01	25.45
DOWN	25.03	25.09	26.5	25.54	24.99	25.82	25.49	25.63

Y.Close	ema20	macd	Mom5	Mom10	Mom20	Roc5	Roc10	Roc20
UP	28.66	-3.382	-4.765	-7.49	-11.18	-0.170	-0.243	-0.3249
UP	28.16	-3.395	-3.568	-6.556	-10.56	-0.132	-0.218	-0.3111
UP	27.82	-3.265	-2.557	-4.461	-8.254	-0.093	-0.153	-0.2507
UP	27.51	-3.135	-0.806	-2.528	-9.497	-0.031	-0.093	-0.279
DOWN	27.38	-2.873	3.109	0.17	-5.711	0.135	0.006	-0.1795
DOWN	27.29	-2.608	3.22	-1.545	-2.88	0.138	-0.055	-0.0982

Annex 2: Adjusted cp table (in terms of α) in one-day window

	α	|T|	$R(T)$	$R_{CV}(\beta)$	$SE_{CV}(\beta)$
1	0.020167	1	0.494835	0.5090999	0.0110874
2	0.009100	2	0.474668	0.5233645	0.0110771
3	0.005903	6	0.438269	0.5090999	0.0110874
4	0.003935	7	0.432366	0.4953271	0.0110887
5	0.003771	11	0.416134	0.4997541	0.0110892
6	0.003443	17	0.386621	0.5012297	0.0110892
7	0.003279	21	0.372848	0.5012297	0.0110892
8	0.002951	24	0.363010	0.5036891	0.0110889
9	0.002705	31	0.342351	0.5036891	0.0110889
10	0.002459	33	0.336940	0.4953271	0.0110887
11	0.002213	42	0.314806	0.4859813	0.0110849
12	0.001968	50	0.295622	0.4923758	0.0110879
13	0.001722	56	0.283817	0.4904083	0.0110872
14	0.001476	60	0.276931	0.5007378	0.0110892
15	0.001312	74	0.256272	0.4987703	0.0110892
16	0.001230	83	0.243974	0.4972946	0.0110891
17	0.001148	102	0.219872	0.4987703	0.0110892
18	0.001107	110	0.210034	0.4987703	0.0110892
19	0.000984	131	0.176094	0.4968028	0.0110890
20	0.000738	185	0.122971	0.4972946	0.0110891
21	0.000656	225	0.091490	0.5017216	0.0110892
22	0.000615	232	0.086572	0.5027054	0.0110891
23	0.000590	236	0.084112	0.5027054	0.0110891
24	0.000492	241	0.081161	0.4938515	0.0110884
25	0.000369	367	0.019183	0.4938515	0.0110884
26	0.000328	372	0.017216	0.4938515	0.0110884
27	0.000246	378	0.015248	0.4938515	0.0110884
28	0.000000	440	0.000000	0.4948352	0.0110886

Annex 3: cp table generated by printcp in five-day window

	cp	nsplit	rel error	xerror	xstd
1	0.03426	0	1.00000	1.00000	0.02406
2	0.01713	1	0.96574	0.97859	0.02401
3	0.01124	7	0.83191	0.96039	0.02397
4	0.00857	11	0.78694	0.89615	0.02376
5	0.00830	12	0.77837	0.89079	0.02374
6	0.00750	17	0.73662	0.87045	0.02365
7	0.00696	24	0.64561	0.86403	0.02362
8	0.00642	26	0.63169	0.86403	0.02362
9	0.00535	29	0.61242	0.84261	0.02351
10	0.00482	38	0.56424	0.84690	0.02354
11	0.00428	44	0.53533	0.83940	0.02350
12	0.00375	59	0.46681	0.82655	0.02343
13	0.00321	65	0.44433	0.81478	0.02336
14	0.00286	80	0.39615	0.81692	0.02338
15	0.00268	89	0.35974	0.79979	0.02327
16	0.00214	104	0.31692	0.80407	0.02330
17	0.00178	153	0.21199	0.78373	0.02317
18	0.00161	157	0.20450	0.76874	0.02307
19	0.00143	175	0.17559	0.77302	0.02310
20	0.00107	181	0.16702	0.77409	0.02311
21	0.00080	300	0.03962	0.77195	0.02310
22	0.00054	308	0.03319	0.77516	0.02312
23	0.00000	370	0.00000	0.77088	0.02309

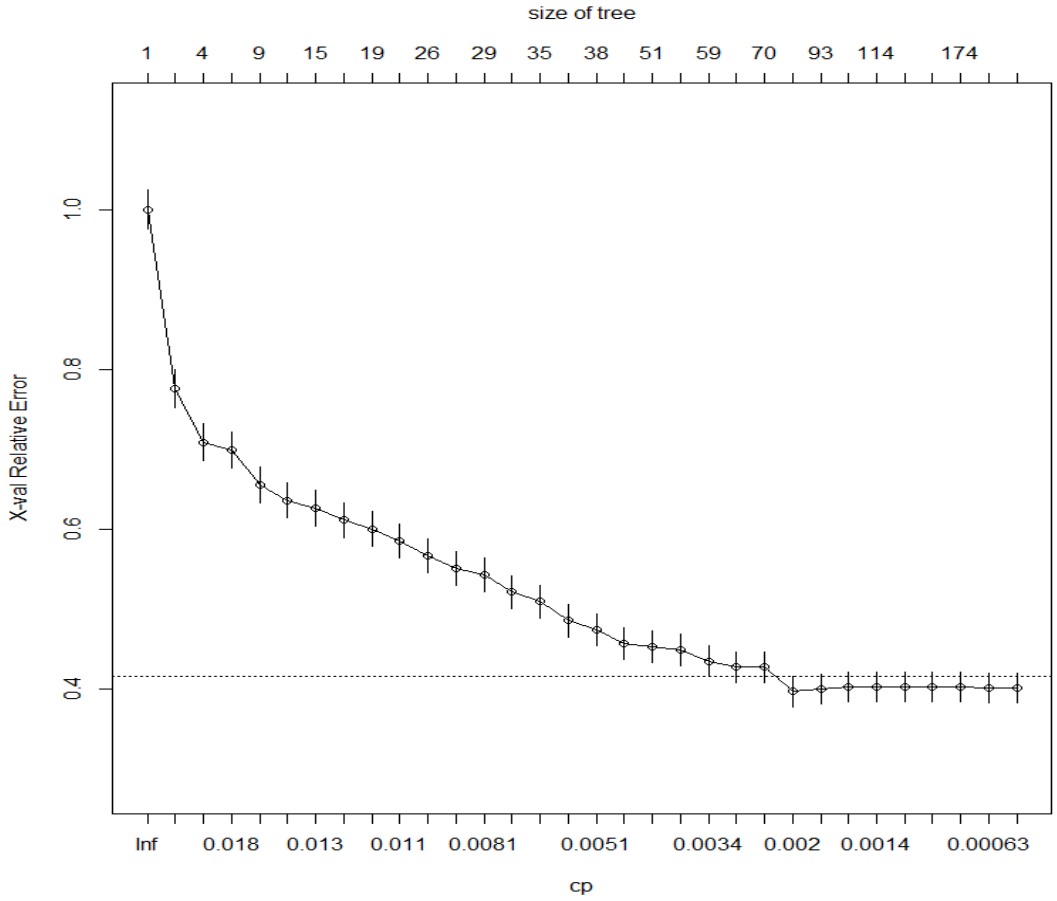
Annex 4: Adjusted cp table (in terms of α) in five-day window

	α	|T|	$R(T)$	$R_{CV}(\beta)$	$SE_{CV}(\beta)$
1	0.01574	1	0.45942	0.45942	0.01105
2	0.00787	2	0.44368	0.44958	0.01103
3	0.00516	8	0.38219	0.44122	0.01101
4	0.00394	12	0.36153	0.41171	0.01091
5	0.00381	13	0.35760	0.40925	0.01091
6	0.00344	18	0.33842	0.39990	0.01086
7	0.00320	25	0.29661	0.39695	0.01085
8	0.00295	27	0.29021	0.39695	0.01085
9	0.00246	30	0.28136	0.38711	0.01080
10	0.00221	39	0.25922	0.38908	0.01081
11	0.00197	45	0.24594	0.38564	0.01080
12	0.00172	60	0.21446	0.37973	0.01076
13	0.00148	66	0.20413	0.37432	0.01073
14	0.00131	81	0.18200	0.37531	0.01074
15	0.00123	90	0.16527	0.36744	0.01069
16	0.00098	105	0.14560	0.36940	0.01070
17	0.00082	154	0.09739	0.36006	0.01065
18	0.00074	158	0.09395	0.35317	0.01060
19	0.00066	176	0.08067	0.35514	0.01061
20	0.00049	182	0.07673	0.35563	0.01062
21	0.00037	301	0.01820	0.35465	0.01061
22	0.00025	309	0.01525	0.35612	0.01062
23	0.00000	371	0.00000	0.35416	0.01061

Annex 5: Adjusted cp table (in terms of α) in 30-day window

	α	|T|	$R(T)$	$R_{CV}(\beta)$	$SE_{CV}(\beta)$
1	0.107231	1	0.454009	0.4540089	0.0110422
2	0.016970	2	0.346778	0.3521889	0.0105936
3	0.008362	4	0.312838	0.3216921	0.0103601
4	0.008116	7	0.287752	0.3172651	0.0103221
5	0.007378	9	0.271520	0.2975898	0.0101399
6	0.005780	10	0.264142	0.2887359	0.0100507
7	0.005657	15	0.235121	0.2843089	0.0100044
8	0.005165	17	0.223807	0.2774225	0.0099299
9	0.004919	19	0.213478	0.2725037	0.0098749
10	0.004673	21	0.203640	0.2656173	0.0097954
11	0.004427	26	0.171667	0.2572553	0.0096947
12	0.003935	28	0.162814	0.2498770	0.0096020
13	0.003443	29	0.158879	0.2464338	0.0095575
14	0.003197	31	0.151992	0.2365962	0.0094257
15	0.002951	35	0.139203	0.2311854	0.0093502
16	0.002459	37	0.133301	0.2203640	0.0091928
17	0.002213	38	0.130841	0.2149533	0.0091107
18	0.001968	42	0.121987	0.2070831	0.0089871
19	0.001722	51	0.103296	0.2056075	0.0089633
20	0.001640	56	0.094442	0.2036399	0.0089314
21	0.001476	59	0.089523	0.1972455	0.0088252
22	0.001230	68	0.076242	0.1938023	0.0087666
23	0.000984	70	0.073783	0.1938023	0.0087666
24	0.000820	90	0.054107	0.1800295	0.0085212
25	0.000738	93	0.051648	0.1815052	0.0085484
26	0.000656	108	0.040334	0.1824889	0.0085664
27	0.000615	114	0.036399	0.1824889	0.0085664
28	0.000492	118	0.033940	0.1824889	0.0085664
29	0.000369	170	0.008362	0.1824889	0.0085664
30	0.000328	174	0.006886	0.1824889	0.0085664
31	0.000246	177	0.005903	0.1819970	0.0085574
32	0.000000	201	0.000000	0.1819970	0.0085574

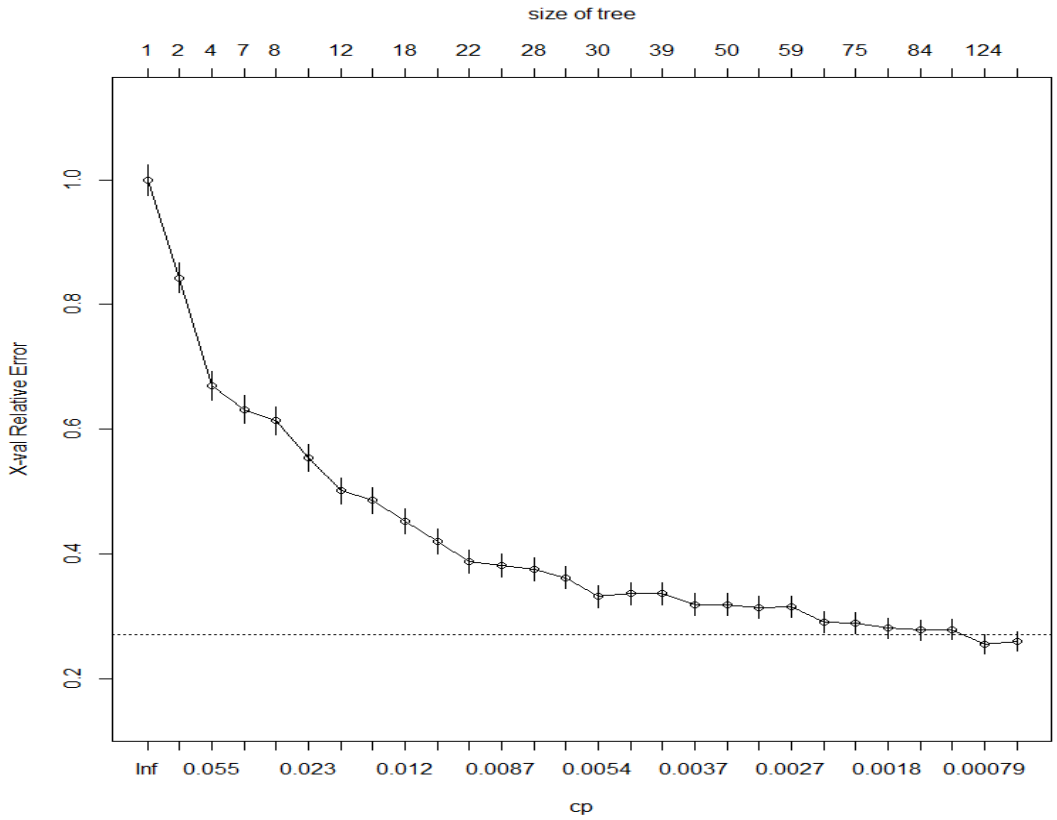
Annex 6: plotcp of T_{max} in 30-daywindow



Annex 7: Adjusted cp table (in terms of α) in 90-day window

	α	|T|	$R(T)$	$R_{CV}(\beta)$	$SE_{CV}(\beta)$
1	0.072307	1	0.441712	0.4417118	0.0110136
2	0.040826	2	0.369405	0.3723561	0.0107218
3	0.014265	4	0.287752	0.2956222	0.0101205
4	0.013773	7	0.244958	0.2788982	0.0099461
5	0.013527	8	0.231185	0.2710280	0.0098581
6	0.007624	10	0.204132	0.2444663	0.0095316
7	0.006231	12	0.188883	0.2213478	0.0092075
8	0.005903	15	0.170192	0.2144614	0.0091031
9	0.005165	18	0.152484	0.1997049	0.0088665
10	0.004919	20	0.142154	0.1854402	0.0086198
11	0.003935	22	0.132317	0.1711756	0.0083538
12	0.003771	25	0.120512	0.1682243	0.0082962
13	0.003443	28	0.109198	0.1657649	0.0082475
14	0.002459	29	0.105755	0.1598623	0.0081279
15	0.002295	30	0.103296	0.1465814	0.0078443
16	0.001968	33	0.096409	0.1485489	0.0078876
17	0.001804	39	0.081653	0.1485489	0.0078876
18	0.001476	43	0.073291	0.1406788	0.0077112
19	0.001312	50	0.062961	0.1406788	0.0077112
20	0.001230	53	0.059026	0.1387113	0.0076659
21	0.001148	59	0.051648	0.1392031	0.0076773
22	0.000984	62	0.048205	0.1283817	0.0074190
23	0.000820	75	0.035416	0.1273979	0.0073947
24	0.000738	78	0.032956	0.1239547	0.0073085
25	0.000689	84	0.028529	0.1224791	0.0072709
26	0.000492	89	0.025086	0.1229710	0.0072835
27	0.000246	124	0.007870	0.1126414	0.0070118
28	0.000000	156	0.000000	0.1146090	0.0070649

Annex 8: plotcp of T_{max} in 90-day window



Annex 9: R code

```
library(quantmod)
getSymbols("BIDU",from="2008-01-01",to="2019-12-31")
#BIDU <- readRDS(file= "RDS_BIDU.rds") #alternative to
getSymbols()
colnames(BIDU) <-
c('Open','High','Low','Close','Volume','Adjusted')

### target variable
window <- 5 # or 1 or 30 or 90
BIDU$change <- (lag(BIDU,k = -window)-BIDU)$Close
BIDU <- BIDU[-c((nrow(BIDU)-89):nrow(BIDU))]
# no NAs for up to 90 days
library(data.table)
Y.Close <- as.factor(ifelse(BIDU$change>0, "UP",  "DOWN"))
treedata <- data.table(Y.Close)
```

```
###feature extraction
library(TTR)
treedata$sma5 <- SMA(Cl(BIDU),n=5)
treedata$sma10 <- SMA(Cl(BIDU),n=10)
treedata$sma20 <- SMA(Cl(BIDU),n=20)
treedata$wma5 <- WMA(Cl(BIDU),n=5)
treedata$wma10 <- WMA(Cl(BIDU),n=10)
treedata$wma20 <- WMA(Cl(BIDU),n=20)
treedata$ema5 <- EMA(Cl(BIDU),n=5)
treedata$ema10 <- EMA(Cl(BIDU),n=10)
treedata$ema20 <- EMA(Cl(BIDU),n=20)
treedata$macd <- MACD(Cl(BIDU), nFast = 12, nSlow = 26, nSig =
4, maType = EMA, percent = FALSE)[,1]
treedata$Mom5<-momentum(Cl(BIDU),n = 5)
treedata$Mom10 <-momentum(Cl(BIDU),n = 10)
treedata$Mom20 <-momentum(Cl(BIDU),n = 20)
treedata$Roc5 <-ROC(Cl(BIDU),n = 5,type = "discrete")
treedata$Roc10 <-ROC(Cl(BIDU),n = 10,type = "discrete")
```

```
treedata$Roc20 <-ROC(Cl(BIDU),n = 20,type = "discrete")
treedata$rsi <-RSI(Cl(BIDU), n=14)
treedata$wad <- williamsAD(BIDU[,c("High","Low","Close")])
treedata$cci <- CCI(BIDU[,c("High","Low","Close")])
treedata$obv <- OBV(BIDU$Close,BIDU$Volume)
treedata$Kprc <- stoch(BIDU[,c("High","Low","Close")], nFastK
= 14, nFastD = 3)[,1]
treedata$Dprc <- stoch(BIDU[,c("High","Low","Close")], nFastK
= 14, nFastD = 3)[,2]
treedata$Wprc <- WPR(BIDU[,c("High","Low","Close")])
treedata <- treedata[-(1:25)] #no NAs
```

```
### training and testing data set
# including specific options in set.seed() for easier
reproducibility
set.seed(7, kind = "Mersenne-Twister", normal.kind =
"Inversion")
rowstraining <- sample(x = 1:nrow(treedata),size =
round(0.70*nrow(treedata)))
training <- treedata[rowstraining,]
testing <- treedata[-rowstraining,]
```

```
### root errors
train.DOWN <- as.numeric(table(training$Y.Close)[1])
train.UP <- as.numeric(table(training$Y.Close)[2])
train.root.error <- min(c(train.DOWN,train.UP))/nrow(training)
test.DOWN <- as.numeric(table(testing$Y.Close)[1])
test.UP <- as.numeric(table(testing$Y.Close)[2])
test.root.error <- min(c(test.DOWN,test.UP))/nrow(testing)
```

```
### growing phase, and pruning phase, cross-validation
library(rpart)
set.seed(16, kind = "Mersenne-Twister", normal.kind =
"Inversion")
maxtree <- rpart(Y.Close~.,
              data=training,
              control=rpart.control(minsplit = 0,
                                    cp=0,
                                    xval = 10),
              method ="class")
CPTABLE.original <- maxtree$cptable
CPTABLE.adjusted <- CPTABLE.original*train.root.error
plotcp(maxtree)
cps <- maxtree$cptable[,1]
alphas <-cps*train.root.error
mincp <-
maxtree$cptable[which.min(maxtree$cptable[,'xerror']),'CP']

### maximal tree characteristics and variable importance
max.VARIMP <- maxtree$variable.importance
max.description <- c(
  nrow(maxtree$frame),# all nodes
  sum(maxtree$frame$var=="<leaf>"), #terminal nodes
  max(rpart:::tree.depth(as.numeric(rownames(maxtree$frame))))
#depth
  )
```

```
### tree with minimal cv-error
min.cv.tree <- prune(maxtree,mincp)
min.cv.VARIMP <- min.cv.tree$variable.importance
min.cv.description <-c(
  nrow(min.cv.tree$frame),
  sum(min.cv.tree$frame$var=="<leaf>"),
max(rpart:::tree.depth(as.numeric(rownames(min.cv.tree$frame))
))
  )

### optimal tree with respect to 1-SE rule
minerror <-
maxtree$cptable[which.min(maxtree$cptable[,'xerror']),'xstd']
mincp.1SE <-
maxtree$cptable[which.max(maxtree$cptable[,'xerror']<min(maxtr
ee$cptable[,'xerror'])+minerror),'CP']
opt.cv.tree <- prune(maxtree,mincp.1SE)
opt.cv.VARIMP <- opt.cv.tree$variable.importance
opt.cv.description <-c(
  nrow(opt.cv.tree$frame),
  sum(opt.cv.tree$frame$var=="<leaf>"),
max(rpart:::tree.depth(as.numeric(rownames(opt.cv.tree$frame))
))
  )
```

```
library(caret)
#performance evaluation
CM.min.train <- confusionMatrix(
  data = predict(object = min.cv.tree,
                 newdata = training,
                 type = "class"),
  reference = training$Y.Close,
  positive = "UP")
CM.opt.train <- confusionMatrix(
  data = predict(object = opt.cv.tree,
                 newdata = training,
                 type = "class"),
  reference = training$Y.Close,
  positive = "UP")
CM.max.train <- confusionMatrix(
  data = predict(object = maxtree,
                 newdata = training,
                 type = "class"),
  reference = training$Y.Close,
  positive = "UP")
CM.min.test <- confusionMatrix(
  data = predict(object = min.cv.tree,
                 newdata = testing,
                 type = "class"),
  reference = testing$Y.Close,
  positive = "UP")
CM.opt.test <- confusionMatrix(
  data = predict(object = opt.cv.tree,
                 newdata = testing,type = "class"),
  reference = testing$Y.Close,
  positive = "UP")
CM.max.test <- confusionMatrix(
  data = predict(object = maxtree,
                 newdata = testing,
                 type = "class"),
  reference = testing$Y.Close,
  positive = "UP")
```